A Reinforcement Learning Approach to Making Decisions On An NBA Court Using A Text Based Basketball Environment

Sharanshangar Muhunthan John Dion
Syed Irtiza Hussain

March 2023

Contents

1 Abstract

Since the origin of the game of basketball, players, coaches, and fans have faced the challenge of making decisions to increase the chances of their team winning. Most often statistics are employed to keep track of player and team performance on a regular basis. These *stats* include points per game, rebounds per game, assists per game, player +/-, PER (player efficiency rating) and more. Given such player statistics, an agent can be used to decide what the player should do in a given situation. For example, player A near the three point line may wish to pass to player B on the three point line who has a better 3-point percentage in order to improve the average number of points won. Reinforcement learning was used to determine the most effective decision in this project.

Reinforcement learning is a machine learning method that relies on rewards and punishments to make the best decision in each state. As an example, consider the game *Flappy Bird*. This game involves a bird and numerous pipes, and the player needs to tap the bird at the correct frequency so the bird goes through the pipe without hitting it. The player gets a point for each pipe the bird successfully traverses through. A reinforcement learning task can be used to teach an agent to maximize the score. In this case, the state can be the bird's current position and whether it's going up or down, as well as the height of the nearest pipe. The agent will be rewarded some positive number of points each time it goes through a pipe and loses some points each time it hits a pipe. We train the agent to maximize the number of points, and with enough time the agent will be able to identify the frequency of taps needed to go through each pipe.

In this project, a similar method was applied to make decisions on an NBA court (from an offensive perspective). Using NBA stats, a text-based basketball game environment was created. Given a set of player's stats, defender stats, team stats and working under the assumption that defence was played man-to-man, numerous episodes were simulated until the agent converged on an optimal policy for making decisions at each state. To achieve this, deep Q-Learning was used [9].

It was shown that high performing players followed the model closely, whereas low performing players tended to stray away from the model's recommendations.

2 Introduction

2.1 Terminology and Definitions

Definition 1 (Agent) *The agent is the learner and decision-maker. The agent is what interacts with the environment and learns based on the feed back of the environment.*

Definition 2 (State) *a representation of the current environment the agent is in.*

Definition 3 (State space) *Set of all possible states, often denoted with* $\mathcal{S}$

Definition 4 (Action space) *Set of all possible actions, often denoted with* $\mathcal{A}$

Definition 5 (Policy) *Often denoted with* $\pi(a|s)$*, is a probability distribution of state to action. Let* s *be a state, and* a *be an action,* $\pi(a|s) = \mathbb{P}(a|s)$*, or the probability that the agent takes action* a *when in state* s*. The optimal policy, denoted with* π_**, is a policy that maximizes the value of all states.*

Definition 6 (Step function) *A function that performs one action and returns the reward observed after taking that action.*

Definition 7 (State-Value function) *Let the set of states be* $\mathcal{S}$*, set of actions* $\mathcal{A}$*, and the set of real numbers be* $\mathbb{R}$*. A state-value function is a function* $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *that maps each state and action to a value. The higher the value, the more favorable it is to take said action when in that state. Often denoted with* q_**.*

Definition 8 (Value function) *A function* $\mathcal{S} \to \mathbb{R}$ *that returns the value of a state when using a specific policy* π*. Often denoted with* v_π*. The higher the value of the state, the more favorable it is to get to that state.*

Definition 9 (Q-Learning) *A method of reinforcement learning that works by predicting and converging to a State-Value function. This is different to other methods that rely on converging to an optimal policy or optimal value function.*

Definition 10 (Back propagation) *A method in machine learning used to optimize a neural network. Back propagation uses residual error of predicted value to adjust weights of a neural network to get a closer prediction value.*

Definition 11 (ϵ-Greedy) *ϵ-greedy is a method of exploration for an agent during its training phase. It is split into 2 phases. The exploitation phase, where it always performs the action it thinks is best. The exploration phase where it takes a completely random action. The exploration phase happens with probability* ϵ*. The exploitation phase thus happens with probability* $1 - \epsilon$*.*

Definition 12 (Activation function) *A function used in artificial neural networks that outputs a small value for small inputs and a larger value for inputs that exceed a threshold. For example, consider the sigmoid activation function* $\frac{1}{1+e^{-x}}$*. Input values that are negative are close to 0, positive input values are close to 1.*

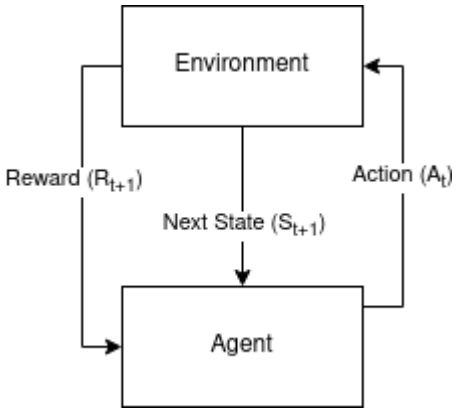
Definition 13 (Residuals) *The difference between predicted value and true value. Let predicted value be* $\hat{y}$ *and true value be* y*. Residual* $r = y - \hat{y}$

Definition 14 (Optimal value function) *The value function under an optimal policy* π*. Often denoted with* v_**.* $v_*(s) = \max_\pi v_\pi(s)$

Definition 15 (Epoch) *Refers to the one entire passing of training data through the algorithm, multiple epochs can be seen as doing multiple sets of a similar exercise. Multiple epochs help the neural network learn. For our implementation epochs were crucial for memory management (doing 600,000 consecutive episodes requires a lot of RAM, doing 200 epochs of 3000 episodes was easier on the computer).*

2.2 Reinforcement Learning in a Nutshell

Reinforcement learning is a machine learning training method in which desired behaviours are rewarded and undesired behaviours are punished. Reinforcement learning makes the agent sense and interpret its environment [1], take actions and decisions, and learn by trial and error. This can be achieved in numerous ways, and with many algorithms. Generally, however, the algorithms involve converging to an optimal policy, value function or state-value function. Evaluating any of those results in an optimal way of making decisions. For example, evaluating the optimal state-value function as in Q-Learning gives us a function that tells us how good or bad each action is in each state, and then one simply chooses the action that yields the largest value. Further, the agent learns by evaluating what reward or punishment is received in the environment after performing an action, and then uses that information to evaluate and converge to a better policy, value function or state-value function. In general, the reinforcement learning process is as follows; first, the agent chooses and performs an action, then the environment responds with the next state after performing the action, and the reward or punishment for doing that action. Then with both the reward and next state the agent learns and improves until it converges on an optimal decision making policy. Historically, reinforcement learning has been used to train self driving cars, industry automation, finance, translation and many other real world problems that can be broken down into a decision process [13][2].

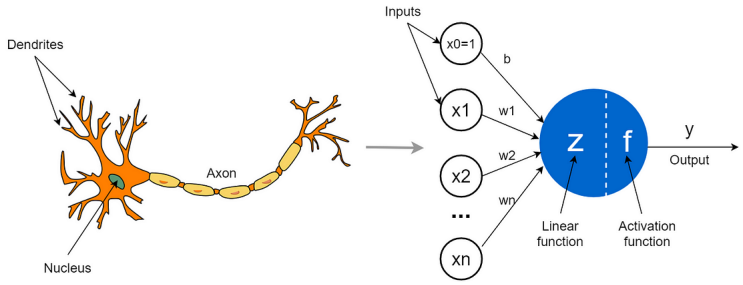


[1] The term *interpret* is used loosely here, in most cases the *interpreter* would do that job and give the agent a vectorized representation of the state, but the term *interpret* is used to show that the agent must *understand* its state in order to make decisions

[2] To be more specific, the processes need to follow a Markov decision process, which can be described as a decision process related to maximizing reward, where the data available is the set of states, set of actions, a probability distribution of states and next states given an action and a reward function of going from one state to another [13]

2.3 Neural Networks in a Nutshell

Artificial neural networks (ANN) is a computational or mathematical model of processing elements that receive inputs and provide outputs based on predefined activation functions. This model loosely models the neurons in a biological brain. Neural networks work as a collection of *perceptrons* which are meant to simulate neurons. Concretely, 1 perceptron takes in many inputs and spits out a binary output [3]. The way it accomplishes this is with an activation function. There are many activation functions, but the most common are sigmoid $\left(\frac{1}{1+e^{-x}}\right)$, RELu $(\max(0, x))$, and arctan $(\tan^{-1}(x))$[4]. The process with a single perceptron is as such:



Layering many perceptrons gives us a neural network, where the output may be multiple instead of just 1. When there are multiple layers, the middle layers are called *hidden* layers. They are called hidden because often there is no interaction with those layers outside of the network itself, and often times there is no way to know the exact weights that are inside, nor is it necessary. The way a neural network is trained is with forward and back propagation. First, a neural network with random weights is initialized. Then with training data you feed the data forward and see the output. Then you calculate the residual and back propogate the error and adjust the weights so that you are closer to the correct answer. You continue this process until the residuals are as low as you want them to be [10].

Neural networks in practice are quite versatile. When they were created they were initially used for classification problems, such as identifying hand written digits and OCR (optical character recognition). However, in modern times creative neural network structures have allowed for more versatile uses such as CNNs (convolutional neural networks) for better image classification, GANs (generative adversarial network) for super resolution and image creation (as in DALL-E) and of course, reinforcement learning. In this project specifically it was used as the *brain* of our agent and predicts a Q-function that evaluates the value of each action.

[3] In practice, it depends on the activation function and is most often not exactly binary

[4] If we are looking at a single perceptron and using a sigmoid activation function, then we essentially have reformatted a logit regression

2.4 Overview of Approach

In order to tackle the problem of decision making on the court, one must first describe what decisions need to be made and in what context. The reinforcement learning agent requires both a state [5] and reward function and with a historical record of both will pick an action [6]. Thus let our state space be $\mathcal{S}$, action space be $\mathcal{A}$ and finally reward function be $\mathcal{R} : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Our state space $\mathcal{S}$ is the following:

- Current player position (x, y)
- Team mate positions
- current player stats (distance per dribble, field goal at midrange, paint, threepoint line, offensive rebounds, etc.)
- defending player stats (steal, block, defensive rebounds, etc.)
- distance from net
- is at three point distance (boolean)
- time on shot clock

The set of actions are:

- Shoot
- Dribble (forward, backwards, to the side, diagonally) one unit (unit is relative to each player)
- Pass to team mate

And rewards as such:

Result	Reward
2 point made	2
3 point made	3
offensive rebound	1.5
free throw made	1
shot blocked	-1.5
ball stolen	-1.5
defensive rebound	-1.5
shot missed	-1.5
otherwise	0

[5] a state is essentially a snapshot of what the playing field currently looks like

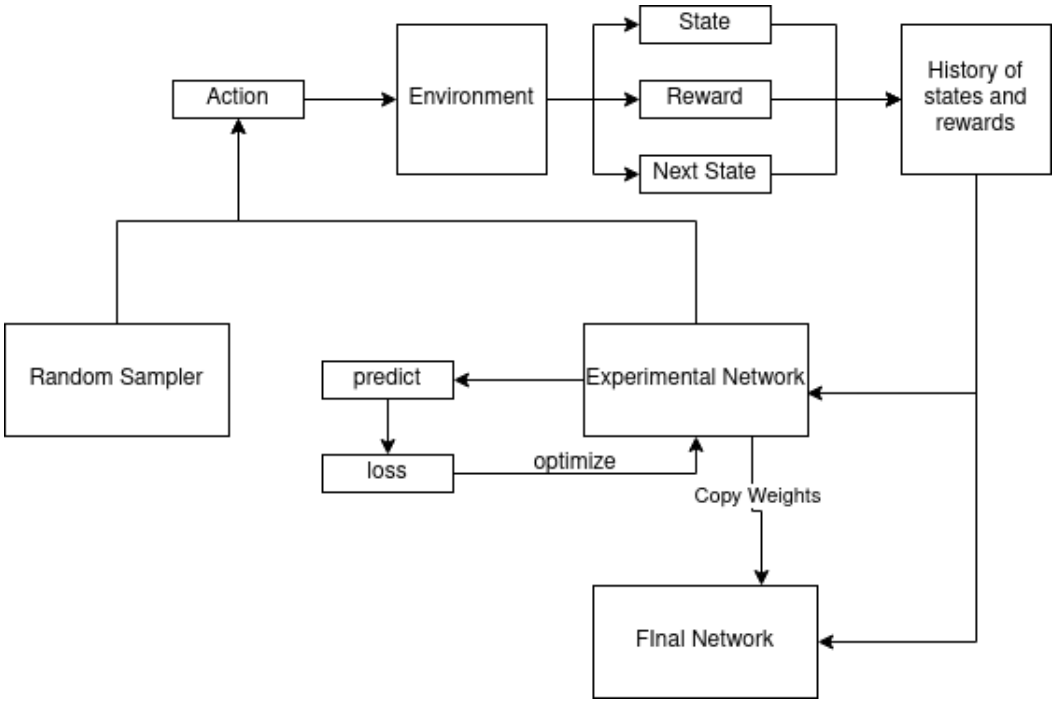
[6] in the case of deep Q-learning, the historical record will be in the form of the neural network's weights

Punishments for turnovers and defensive rebounds were estimated with a ratio of points off turn overs by number of turn overs, since there was no significant correlation between blocks, steals or defensive rebounds and number of points scored (more on that in section 3.4).

Deep Q-learning (using a neural network) was the approach taken due to the size of the state. One state vector had more than 130 elements it was keeping track of, thus making a discrete table with every possible state we came across was unrealistic.

To train the model, an ϵ-greedy initial approach is taken [7]. Then a data set of current state, next states and rewards is made. This data set is then used to optimize the model. A more detailed explanation follows in section 4.

Thus, the entire process looks like [11]:



[7] An ϵ-greedy approach is an approach where the agent picks the best known action at each step, but with probability ϵ pick a completely random action instead. This approach is used to promote exploration of the agent, since it may think it found the best move, but after taking a random action finds that the random action was indeed better.

3 Environment

The environment was chosen to be text-based due to ease of creation. In an ideal world, access to the code of an NBA2K game would have been the most *efficient* way of approaching this problem, since the stats used are already accurate for each player, and they have far more complicated states such as fatigue, automatic movement of other players while idle and more which were were not tackled in this environment [8]. Further, there are examples of text based environments with random events being able to effectively train reinforcement learning agents as in Microsoft's TextWorld [6]. However, unlike TextWorld which relies on randomly generated events, this environment was made as statistically accurate as possible by using what stats were made publicly available by the NBA, such as field goal percentages, pass completion rates and others[2]. Essentially, based on scraped data, player objects were created. The environment then (based on action selected) utilizes the objects created and statistics available to finally predict whether that action was a success or not. For example, if the agent chooses to shoot from the three point line, the environment will find the shooting players three point percentage, and then flip a coin (note that the coin would be unfair, since the probability of making a three pointer would in all likelihood not be 50%). If the coin lands on heads, we consider the shot made, otherwise failed. If the shot is made, the environment spits out a reward of 3 points. If the shot was blocked, spits out a reward of -1.5. Of course more considerations were made for each action such as getting fouled, but in the subsections below go in more detail about each.

Further, for the task at hand, an episodic method was chosen. In this case, each episode has a 24 second limit (shot clock). An episode continues until either a shot is made or the ball is turned over. When a shot is made, its respective reward is given. When turned over, its respective punishment is given (see reward table).

3.1 Shooting

NBA had statistics available for shots beyond the three point line, at mid-range, in the paint or in restricted area. However, nothing was available for exact (x,y) positions. Thus, to evaluate field goal percentage, the solution was to just see what area the player was in and use the respective statistic. For example, if the player was standing in mid range, then the mid range field goal would be used.

[8] An interesting aside, Max Holloway (a UFC fighter) was quoted on using the UFC video game to try out different moves and tactics in a fight against his future opponent, and with great success[5]. This may show that video games (if made accurately) may help with the tactical aspects of sports

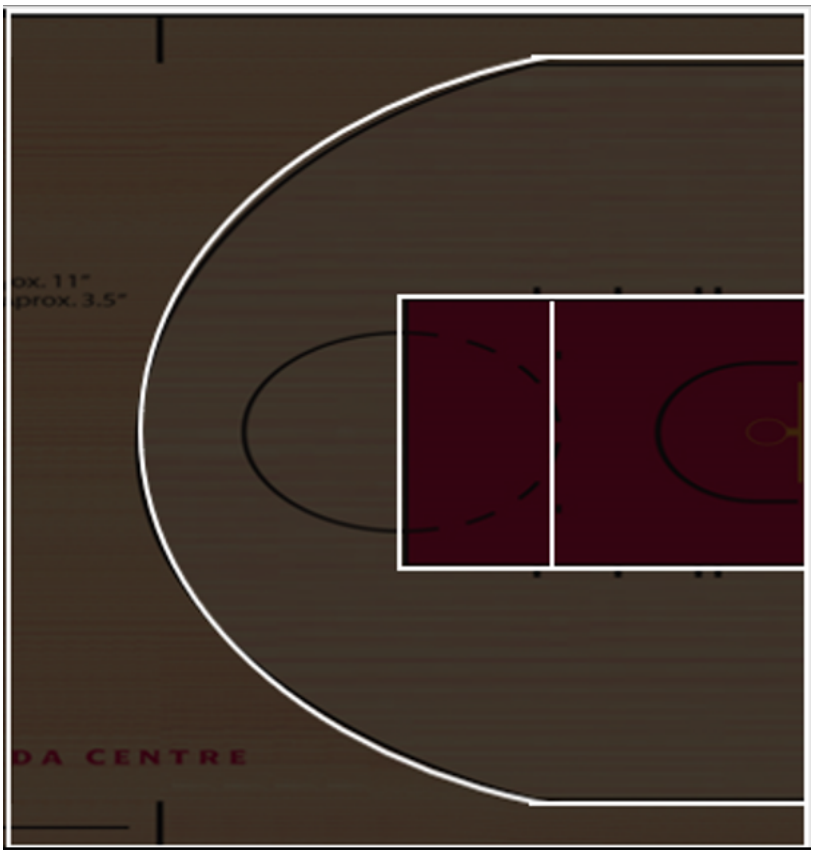


Figure 1: The half court divided into the shooting ranges (going from furthest from net to closest), three point range, mid range, in the paint and restricted area

One thing that was evaluated but later was found insignificant was the effect of different variables on field goal percentage. One such metric was shot clock time [9]. However, the model found no significant correlation between shot clock and field goal percentage (see Table 3). Most R^2 results were extremely small, showing almost no significant correlation between shot clock and field goal percentage.

3.2 Passing

Passing ability is another key factor the model needs to account for. Some NBA players are naturally better passers than they are shooters, and their

[9] wanted to evaluate the quality of being "clutch", in essence, see if there was a possible regression formula that can be used to increase or decrease a shooters base field goal percentage based on time left on the shot clock

vision on the court allows for more efficient ball movement and sets up other players for more favourable shots. NBA Court Optix had passing data available between 11 regions they specified. These were the regions used to evaluate passing completion[10].

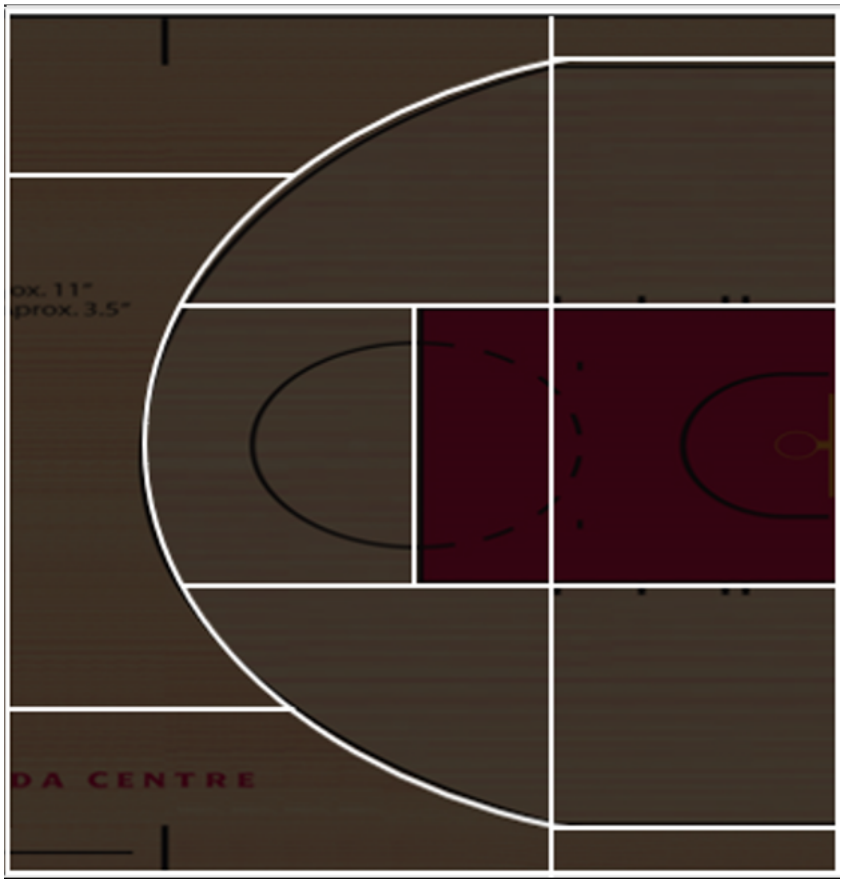


Figure 2: The 11 passing regions as offered by NBA Court Optix

The NBA did not have the in depth data available for what was required for the model, so some improvisation was needed. Because the model was so dependent on the location of players on the court, the general passing efficiencies of the team from an arbitrary region A would not suffice. Let $\mathbb{P}(\mathcal{P}_{i,j})$ be the probability a pass is completed from region i to region j, and $\mathbb{P}(\mathcal{P}_i)$ be the probability of completing a pass from region i. One could not simply set

[10]This is different from the 4 shooting regions. However, since we tracked exact (x,y) coordinates of each player, evaluating stats for passing vs shooting essentially boiled down to which region the point fell in and using that respective statistic. For example, if the player was in an arbitrary passing region i, and was also in the midrange shot region, we would use the pass completion statistics from region i and shot statistics from midrange

$\mathbb{P}(\mathcal{P}_{i,j}) = \mathbb{P}(\mathcal{P}_i)$, because the probability of one is larger than the other, since one contains the event of the other [11]

A team-by-team passing network displayed the passing completion percentage from each region on the court, while also showing assist percentage from one specific area of the court a to another, b [3]. Let $\mathbb{P}_{a,b}(\mathcal{A})$ be the probability of a given pass being an assist and $\mathbb{P}(\mathcal{S}_i)$ be the probability of a shot being made from a specific zone i.

From this, assuming $\mathbb{P}(\mathcal{P}_{a,b})$ and $\mathbb{P}(\mathcal{S}_b)$ are essentially independent, one can derive the equation:

$$\mathbb{P}_{a,b}(\mathcal{A}) = \mathbb{P}(\mathcal{P}_{a,b} \cap \mathcal{S}_b) = \mathbb{P}(\mathcal{P}_{a,b})\mathbb{P}(\mathcal{S}_b) \tag{1}$$

Rearranging, one arrives at:

$$\mathbb{P}(\mathcal{P}_{a,b}) = \frac{\mathbb{P}_{a,b}(\mathcal{A})}{\mathbb{P}(\mathcal{S}_b)} \tag{2}$$

Note that if $\mathbb{P}(\mathcal{S}_i) = 0$, then $\mathbb{P}(\mathcal{P}_{a,b}) = 0$.

Of course, this does not account for passes that do not immediately result in a shot attempt, whether that be the receiving player dribbling the ball to another region of the court or passing it again to a different player. To somewhat counter this, and mitigate the influence of regions where the assist percentage is low, the weighted average is taken of 1 and 3, where $\tilde{\mathcal{P}}_{a,b}$, $\mathcal{A}_{a,b} \in \mathbb{N}$ denote the total number of completed passes and total number of assists respectively from zone a to b:

$$\mathbb{P}(\mathcal{P}_{a,b}) = \frac{\mathcal{A}_{a,b}\dfrac{\mathbb{P}_{a,b}(\mathcal{A})}{\mathbb{P}(\mathcal{S}_b)} + (\tilde{\mathcal{P}}_{a,b} - \mathcal{A}_{a,b})\mathbb{P}(\mathcal{P}_a)}{\tilde{\mathcal{P}}_{a,b}} \tag{3}$$

Again, if $\mathcal{P} = 0$, then $\mathbb{P}(\mathcal{P}_{a,b}) = 0$. I.e. if historically, there are no passes made from region a to region b, then the environment considers the probability of that pass being made as 0 [12].

3.3 Dribbling

Dribbling is an important component that was investigated to make the environment as statistically accurate as possible. It's essential to measure the time and number of dribbles it takes for a player to move from their starting position to another position, when an arbitrary direction vector is applied to their starting position.

The equations required a lot of factors to be considered such as a player's average speed, average number of dribbles per second and average stride length by position to ensure accuracy (see Table 1). The NBA did not have data on

[11] The probability of making a pass from region a would also count the probability of making a pass from region a to region b. More clearly, $\mathbb{P}(\mathcal{P}_a) = \mathbb{P}(\cup_i \mathcal{P}_{a,i})$

[12] In practice, the data only has a frequency of 0 passes for passes that would, in reality, be a *bad* pass to make, for example trying to pass from the left corner three to the right.

players stride lengths, to account for stride length we will assume 1 stride = 1 dribble. The equations used within the function to calculate the time and number of dribbles are:

$$\text{Time to move} = \frac{\text{Distance Travelled}}{\text{Average Player Speed}} \tag{4}$$

$$\text{Distance Travelled} = \text{Time to move} \times \text{Average Player Speed} \tag{5}$$

$$\text{Number of dribbles} = \text{Time to move} \times \text{Dribbles per Second} \tag{6}$$

Equations (4) and (5) are based on the fact that speed $= \frac{\text{distance}}{\text{time}}$.

The function used in the simulations takes three arguments: the player's starting position, the direction vector and the position. Using the previously stated information and the three arguments, we are able to get the distance travelled per dribble, the time it took to accomplish that dribble and the number of dribbles taken. These were necessary for simulating the act of a dribble to find the new position after dribbling, and how much time passed after dribbling (dribbling when there is less than 1 second left on the shot clock would not be ideal).

3.4 Turn Overs

The environment treats a ball lost as a turn over[13]. Thus, if there is a block, steal or defensive rebound the environment treats it as a turn over and ends the episode. The punishment for a turn over was set to -1.5. This is the approximate average ratio of points off turn overs to turn overs (per 100 possessions). Expected number of points gained per turn over was not used because there was no significant correlation (see Table 3). Other types of turn overs that are not punished are of course when the ball is given back to the other team after a shot or a free throw is made. In both cases the reward depends on number of shots made (1 point per bucket).

3.5 Interaction

The way the environment works step by step is that it first gets initialized with a `start()` function. This function will initialize all players in random positions on the court, then return the initial state with the information highlighted in Section 2. Then the agent will pick some action from the available actions, either at random or from its neural network. The action is passed to a `step` function, that performs the action and sends back an observed reward. Then

[13] i.e. ignores cases of a block going back to an offensive player because of how rarely they happen

we query for the current state (after taking action), and repeat the process until the episode terminates (see the environment process figure in the appendix)

Also, at each "step", since it is text based, messages are outputted. One sample message taken during the training process[14]:

```
O.G. Anunoby has ball!
O.G. Anunoby passes to Gary Trent Jr. in box 1
O.G. Anunoby makes pass!
Gary Trent Jr. has ball!
Gary Trent Jr. dribbles BACKWARDS!
Gary Trent Jr. has ball!
Gary Trent Jr. dribbles BACKWARDS!
Gary Trent Jr. has ball!
Gary Trent Jr. dribbles BACKWARDS!
Gary Trent Jr. has ball!
Gary Trent Jr. dribbles FORWARD!
Gary Trent Jr. has ball!
Gary Trent Jr. dribbles FORWARD!
Grayson Allen fouls!
Gary Trent Jr. attempts free throws!
Free throw made!
Free throw made!
Free throws done!
```

[14]It is interesting to see how the agent learns. During this run, the agent realized that it has a high likelihood of getting fouled, and a high likelihood of making a free throw, and thus kept dribbling until it got fouled. This was later adjusted so that the probability of getting the ball stolen increased with each dribble to offset the probability of a foul. Nevertheless, it shows that the agent will find *creative* ways of increasing the number of points.

4 Algorithm

Deep Q-Learning was the methodology used to make the best decisions. Let the set of states be $\mathcal{S}$, and set of actions $\mathcal{A}$. Let $q_* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a function that maps each state and action to its value $\in \mathbb{R}$ [15]. By knowing what q_* is, we can determine what the best action to take for a state s is by taking $\arg\max_{a \in \mathcal{A}} \ q_*(s, a)$[13]. Deep Q-Learning uses a neural network to estimate q_* by using the following:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{7}$$

It has been shown that $Q(s_t, a_t)$ converges almost surely with q_* as $t \to \infty$[13]. To actually predict this function, one first initializes two neural networks Q_e and Q_f, as well as a probability ϵ. To start, to promote exploration, ϵ would be set to a value close to 1, then as we iterate through episodes, decay epsilon by some function δ_ϵ. The goal is to decrease exploration as we increase the network's accuracy. Q_e represents an experimental network, whose weights we will copy over every k steps to Q_f. After initializing the networks, using Q_e we experiment with the environment and collect some sample data. The sample data will consist of the current state, the action taken, the next state and the observed reward. Then with some batches of this sample data, optimize Q_e. By knowing the current state, action taken and reward upon action, we can utilize an error function to fix the networks current rating of each action in each state. Then every k steps, copy over all the weights from Q_e to Q_f. The use of two networks is to establish and ensure stability of our agent.

Optimization is done using a stochastic gradient descent algorithm [16] with a mean squared error-esque loss of the following form that is derived by taking parts of equation (6):

$$\mathcal{L}(s_t, a_t) := (Q(s_t, a_t) - (r_{t+1} + \gamma \max_a Q(s_{t+1}, a)))^2 \tag{8}$$

For this task, the number of collection steps was set to 3000, and instead of waiting for convergence to 0, did 200 epochs of training (200 sets of 3000 episodes, this was chosen because the errors converged to near 0). Further, ϵ was set to 0.9, and $\delta_\epsilon = -(0.1 \times \frac{n}{375})$. δ_ϵ was chosen this way so that at the end of the episode ϵ was still around 0.1, leaving a small chance of discovering new actions. To find this, first one solves the following $\epsilon_0 - (\delta \times \frac{n}{x}) = \epsilon_f$ for x, where ϵ_0 is the starting epsilon (in our case 0.9), δ is the decay rate (we chose 0.1), n is the number of episodes (in our case 3000) and ϵ_f is the desired final epsilon (we chose 0.1). This left us with $x = 375$. From here its clear that δ_ϵ is represented in the $-(\delta \times \frac{n}{x})$ of the aforementioned equation. Thus after finding x, we have our δ_ϵ as mentioned before. The number of steps we copy over the

[15] the higher the number the more favorable it is to take action a when in state s

[16] stochastic gradient descent is a methodology that walks along the gradient of the error function until it hits a minimum, rather than find a actual solution, as is done with linear regression

weights, k was set to 1500. For actually constructing the neural network as well as train knowing the loss, python's `pytorch` package was used.

Thus, the algorithm for training a Q-learning agent is as follows [11]:

Algorithm 1 DQNAgent

Require: Q_e, Q_f, ϵ
  while not converged **do**
    for number of collection steps **do**
      $s \leftarrow$ current environment state
      $e \leftarrow U(0, 1)$
      if $e < \epsilon$ **then**
        $A \leftarrow$ random action
      else
        $A \leftarrow \arg\max_a Q_e(S, a)$
      end if
      $S', r \leftarrow$ Observation when taking action a
      Store (S, A, r, S')
      $\epsilon \leftarrow \epsilon - \delta_\epsilon$
    end for
    Optimize Q_e using Q-value evaluated with stored tuples
    Copy weights from Q_e to Q_f every k steps
    Reset ϵ
  end while

5 Discussion and Conclusion

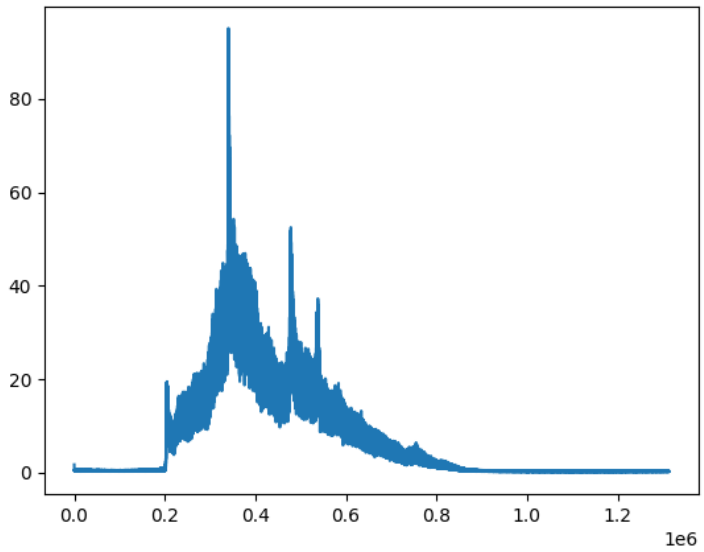


Figure 3: A plot of the loss of the neural network over training

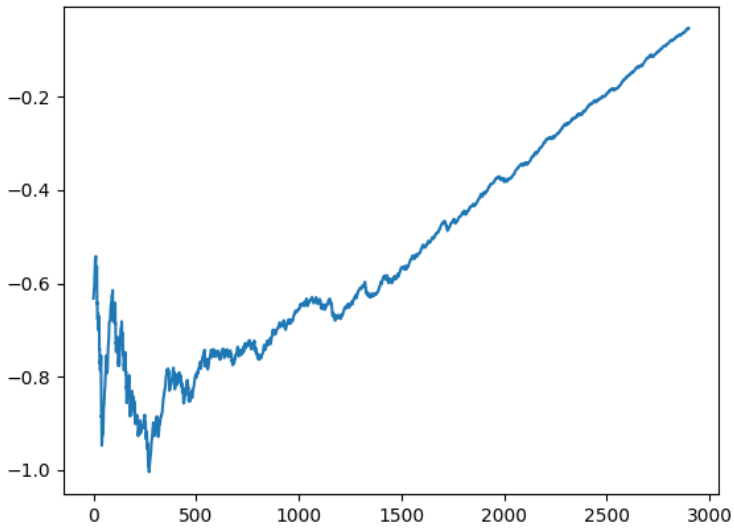


Figure 4: A plot of the average reward in the final epoch

5.1 Training

As one can see on the figure in the left, over the 600,000 episodes, we do have a loss convergence to near 0. The major spikes were due to random exploration. Essentially the spikes tell us that at one point the agent *knew* what the best action was, but upon some exploration, realized that it did not know what the best action was and thus corrected itself. After all the episodes, the error converged to 0 even during the random exploration [17] telling us that it finally converged to some accurate function of evaluating an action in a given state. To further show this, if we look at the second figure of the reward plot, we can see as the episode number approaches 3000, the average rewards increase. Again, this is because as the number of episodes increases, ϵ decreases, meaning we use the network more and more. Thus the increase in reward on average shows that the network performs better than randomly choosing an action.

[17] Since random exploration spikes every 3000 episodes, due to ϵ being set to 0.9 at the start of each epoch, and the flat tail at the end remains low for more than 3000 episodes

5.2 Insights

To evaluate the validity of the model, it was tested on five players from the Toronto Raptors as well as the top three and bottom three teams in the NBA. As the episodes were running for each player; the player's states, the best action the model took, and the (x,y) coordinate the model took it from were noted to generate heat-maps and shot charts to further analyze the validity of the model. For example, Scottie Barnes heat-map (Figure 6) shows that his probability for maximum reward was high when an action was taken in the bright red area. This means that majority of his shots were successful from that location, and passes from there led to a sequence of successful plays. However, the light blue areas were where his probability of maximum reward was low relative to the bright red areas. Comparing this to his shot-chart (Figure 5), it can be seen that the majority of the shots that Barnes made aligned with the high probability of maximum reward position on his heat-map. The shots he missed are visible in the light purple areas, indicating that he should not take shots from those positions. This relatively aligned with his real life shot-chart from 77 games this season, where he made 33.3% from that position. Another player whose heat-map was interesting was Pascal Siakam (figure 13) as we could not generate a shot-chart for him. He tended to frequently dribble left, right, or straight into the paint before ultimately making a pass to OG Anunoby. The model was mimicking his real life style of play as looking at his passing network on NBA CourtOptix, it can be seen that majority of his passes from his starting position (within the model) and the paint were mostly directed towards OG Anunoby. This further validates our model as the model seems to be mimicking the player real life style of play. Since the model looks for the best possible decision to maximise reward, it raises the question; were players/teams whose in real life shot-charts/heat-maps followed those of the models, paid more and had more wins than losses? The top three best teams (Celtics, Bucks, 76ers)(14) and bottom three worst teams (Pistons, Rockets, Spurs)(20) in the NBA were used to test this theory. It was found that the bottom three teams shot-charts, majority of the times went against their real life shot-chart and vice-versa. For example, the Detroit Pistons Isaiah Stewart was going against our model and not shooting as often from mid-range whereas our model predicted that 97% of the shots (14) he would take from that position would maximise the reward. This tells us that Isaiah Stewart should shoot from that position as probability of maximum reward is high. Another example is San Antonio Spurs Keldon Johnson, who scored 67% of his mid-range shots with our model (24) whereas his NBA shot-chart this season, he has significantly low amount of shots from there (25). These three bottom teams seem to have a low correlation with their official shot-charts and our models shot-charts indicating the reason for their high loss record. The top three teams in the NBA seemed to follow our model relatively well. For example, Milwaukee Bucks Brooke Lopez was making 42% (16) of his shots from mid-range which closely followed his official NBA shot-chart of 46.7% shots made from mid-range (17). Boston Celtics Jayson Tatum was averaging 52% shots made from mid-range (18) which again closely followed

his official NBA shot-chart from that position with 48.8% shots made from mid-range (19). This means the model thought that these areas were the best places for the players to shoot to maximize the reward and this mimicked their exact play-style in real life. It seems that our models favoured high performing teams rather than low performing teams. The teams that followed our models had more wins than losses and the opposite for the ones that didn't. Upon further investigation, it was found that players who followed our model received significantly higher pay than those who did not as shown in the table below. For example, Jayson Tatum has a salary of 30 million USD whereas Isaiah Stewart has a salary of 3.4 million USD.

Table 1: Relationship Between Player Salaries (22/23) and our Model

Followed Model	Salary (USD)	Went Against Model	Salary (USD)
Jason Tatum	$30,351,780	Isaiah Stewart	$3,433,320
Pascal Siakam	$35,448,672	Keldon Johnson	$3,873,024
Brooke Lopez	$13,906,976	Kevin Porter	$3,217,631

[1]

All this validates our model and may show that players and teams whose real-life shot-charts/heat-maps followed those of the models were getting paid significantly more and their teams had significantly more wins than losses this season.

5.3 Issues

Of course, this model does have it's shortcomings due to a few factors. The first being data availability. The entire simulator was built on publicly available data which was, unfortunately, quite generic. For example, if a player was just forward of the three point line, the shot would have the field goal percentage of a mid range shot, when the difficulty of it is in fact quite similar to a three pointer. Ideally, a field goal percentage of each (x,y) or by distance from net would have been the better metric to use. Further, because of data availability, for each statistic used, the most granular level data was used as the deciding probability of the success of each action. This caused a few discrepancies. For example, for shot completion, we only had data for midrange, three point, in the paint and in restricted area. This gives us 4 shooting regions. For passing however, NBA Court Optix offers passing data for 11 regions across the half court. To combat this discrepancy in data, we simulated each action with exact (x,y) coordinates. With exact (x,y) coordinates, it was a matter of just querying which region the point was contained in and using its respective statistic. Another issue faced was training time. The model did converge after numerous episodes, however, there were restrictions placed which reduced the overall generality of the model. For example, in all simulations, the guards started past the three point line, and the forwards as well as centers started somewhere in the mid-range to paint area. Thus, the generality of the model depends on whether or not at some point the model tried going towards a different region on the court and found

success. Another issue was the lack of consideration of external forces on the current state, such as fatigue, player's dribbling skill, team mate position IQs and many more. Taking these metrics into account may have promoted play styles that relied more on passing and moving the ball than to set up shots as the model currently converged to. These external forces also give rise to other aspects of the game such as the best time to make a substitution, what the best positioning of each player may be, who would do better to start and more. These considerations give rise to more in depth play making and decision making from both a player and coach perspective. With more computing time and a more detailed environment however, one can add more generality to the states and train the agent longer, thus allowing the model to find more tactics than what it currently has found.

5.4 Future Improvements

In an ideal world, having access to a plethora of very specific data would have helped greatly with training and simulating game situations, which would allow more generality and freedom of decision making for the agent. Likewise, one of many future improvements that can be made to the model were expressed in the Environment section. Using a preexisting and accurate environment of an NBA video game would give rise to the considerations missed above, since the game already takes these into consideration, being fatigue, movement of other players and more. Another improvement that can be made is to use distance based shot metrics instead of region based. That data was available, but was not used in the model. Theoretically however, since we track players by each (x,y) point on the court, it is indeed possible to find the distance to net as well as differentiate between each region (beyond three vs above three). Another improvement that can be made is increasing the generality of other aspects of the model such as moving players. The movement of other players can also theoretically be determined by the model as well, however it would require more training due to the large number of possibilities. Overall, the improvements all are related to increasing the generality and statistical accuracy of the model to allow the model more *creativity* in its solutions. Another main improvement is to audit the reinforcement learning algorithm used, and perhaps choose one that better tackles the problem of sparse and stochastic rewards. Since in this case, it takes quite a number of chosen actions and there is a non deterministic way of receiving positive or negative reward, since it depends on the players statistics, and even then there is a probability of the reward going either way. Thus, picking a better algorithm leads to a model that better converges on a result. There is a lot of recent work that tackles these issues. Such as the LOGO algorithm which was demonstrated during ICLR 2022 [12] which was built around training an agent with sparse reward when there is a sub-optimal policy available to learn from. Or UCSG (upper confidence stochastic game algorithm) [14] that generates confidence sets to pick the best model and performs well for stochastic models, in fact it was designed to tackle problems involving

markets, and board games [18]. There is also the Stochastic reward machine type algorithms presented during AAAI 2022 conference (performed well in a harvesting environment where rewards are both sparse and random) [8] as well as published by Dohmen et. al. [7] in 2021 that tackles sparse and stochastic rewards. In general, as more algorithms solve the well known problem in reinforcement learning related to training an agent with sparse and stochastic rewards, one can improve the underlying methodologies used to train the agent and can thus formulate even more interesting and accurate insights involving making decisions. Many of these algorithms are extremely complicated to digest, but do have applicability in this space of making decisions in a live and random environment, as shown where some of these algorithms perform well with handling problems in markets and board games which rely on numerous random decisions from multiple parties, and thus would be an interesting problem to research and further try to understand in the realm of sports and play making therein.

[18] but one flaw is that the games must be zero sum, but it allows us to train two opposing teams at the same time since it is designed for multiple agents, thus may provide different insights into the ideal way to play from both sides of the court

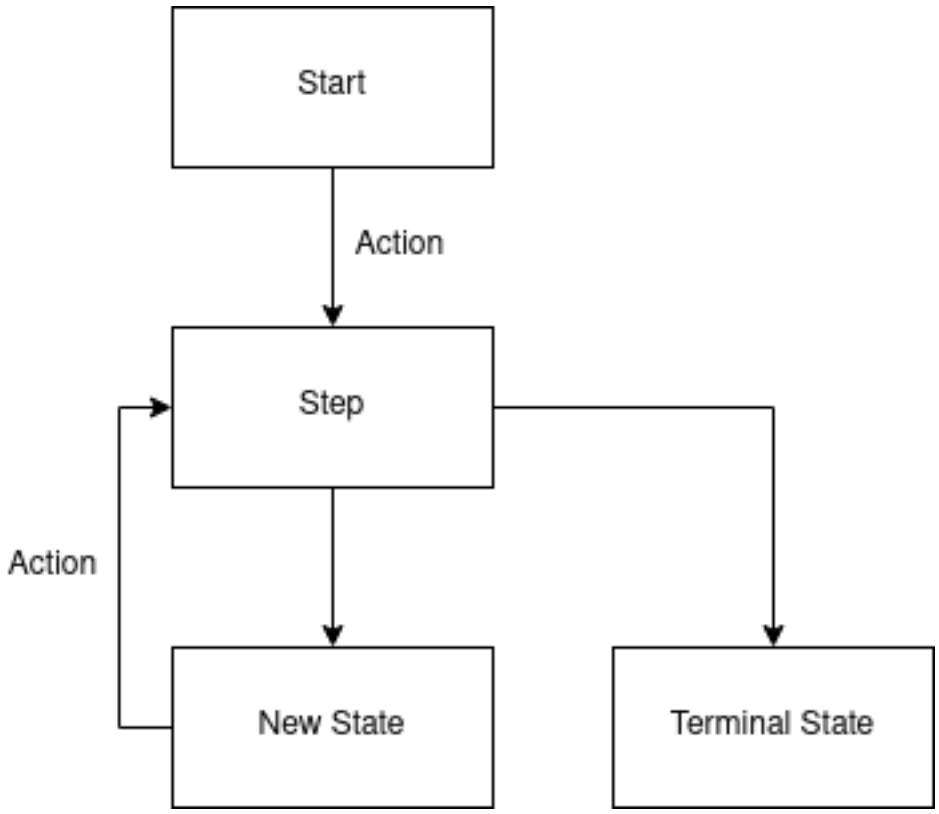
6 Appendix

6.1 Dribbling Data

Table 2: Player Speed and Dribbling Statistics by Position

Position	Avg. Speed (ft/s)	Avg. Dribbles per Second	Avg. Distance per Dribble
PG	4.53	1.37	3.27
SG	4.53	1.37	3.27
SF	4.47	2.16	2.21
PF	4.47	2.16	2.21
C	4.36	3.18	1.42

6.2 Environment Interaction Process



6.3 Regression Results

Table 3: Turnover Regression

	Dependent variable:		
	PTS		
	(1)	(2)	(3)
STL	−0.799		
	(0.660)		
BLK		0.282	
		(0.662)	
DREB			0.606
			(0.294)
Constant	120.354*	113.199	94.520
	(4.852)	(3.119)	(9.708)
Observations	30	30	30
R^2	0.050	0.006	0.132
Adjusted R^2	0.016	−0.029	0.101
Residual Std. Error (df = 28)	2.727	2.788	2.607
F Statistic (df = 1; 28)	1.466	0.181	4.251
Note:			*p<0.1; p<0.05; ***p<0.01

Table 4: Shot Clock Regression Results

	Dependent variable: Field Goal							
	FG	FG2	FG3	FG	FG2	FG3	FG2	FG3
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Interval	2.554	3.165	−1.921	2.207	2.774	−1.933	4.052	−0.723
	(0.379)	(0.435)	(0.450)	(0.628)	(0.666)	(0.685)	(0.481)	(0.479)
Constant	39.650	40.972	32.879	49.212	50.315	23.609	34.897	31.242
	(1.452)	(1.669)	(1.725)	(2.424)	(2.572)	(2.648)	(1.820)	(1.813)
Observations	1,055	1,055	1,055	407	407	407	994	994
R^2	0.041	0.048	0.017	0.030	0.041	0.019	0.067	0.002
Adjusted R^2	0.040	0.047	0.016	0.027	0.039	0.017	0.066	0.001
Residual Std. Error	20.771 (df = 1053)	23.885 (df = 1053)	24.687 (df = 1053)	21.259 (df = 405)	22.556 (df = 405)	23.220 (df = 405)	25.156 (df = 992)	25.070 (df = 992)
F Statistic	45.461 (df = 1; 1053)	52.831 (df = 1; 1053)	18.209 (df = 1; 1053)	12.372 (df = 1; 405)	17.359 (df = 1; 405)	7.951 (df = 1; 405)	70.948 (df = 1; 992)	2.274 (df = 1; 992)

Note: p<0.1; p<0.05; p<0.01

6.4 Shot-charts and Heat-maps

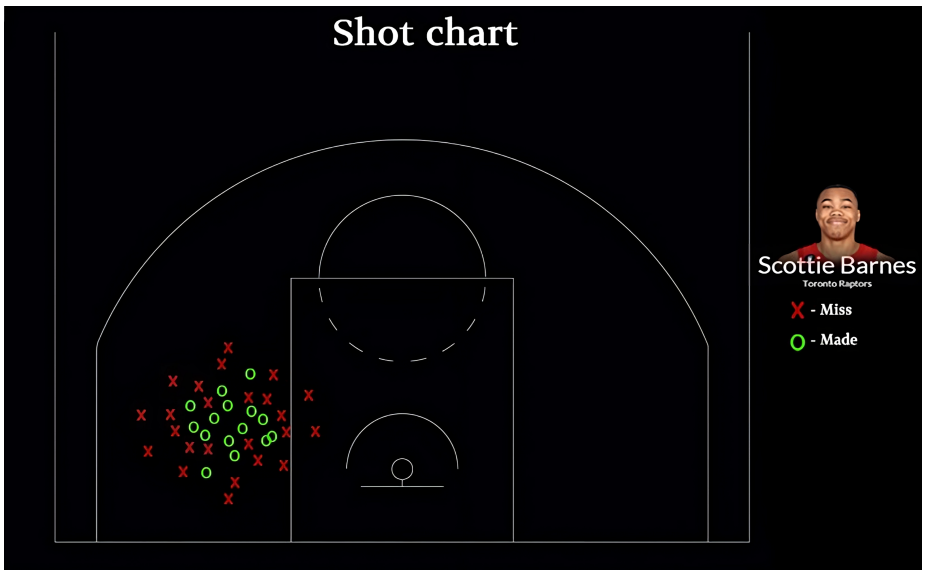


Figure 5: Scottie Barnes Shot Chart

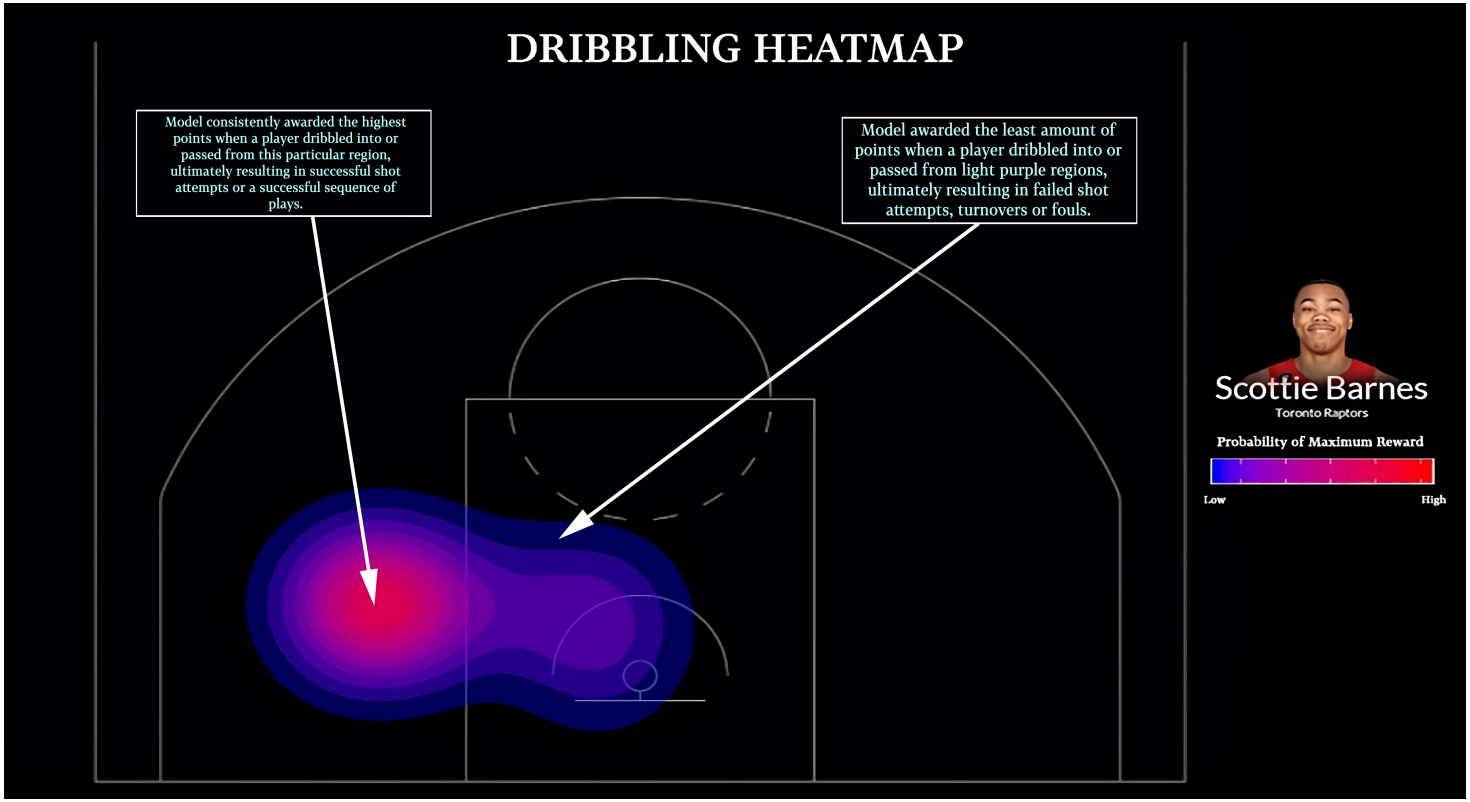


Figure 6: Scottie Barnes Heat-Map

Bright red area: Model consistently awarded the highest points when a player dribbled into of passed from this particular region, ultimately resulting in a successful shot attempts or a successful sequence of plays.
Light purple area: Model awarded the least amount of points when the player was in those positions, plays would ultimately lead to turnovers, fouls, and failed pass/ shot attempts.

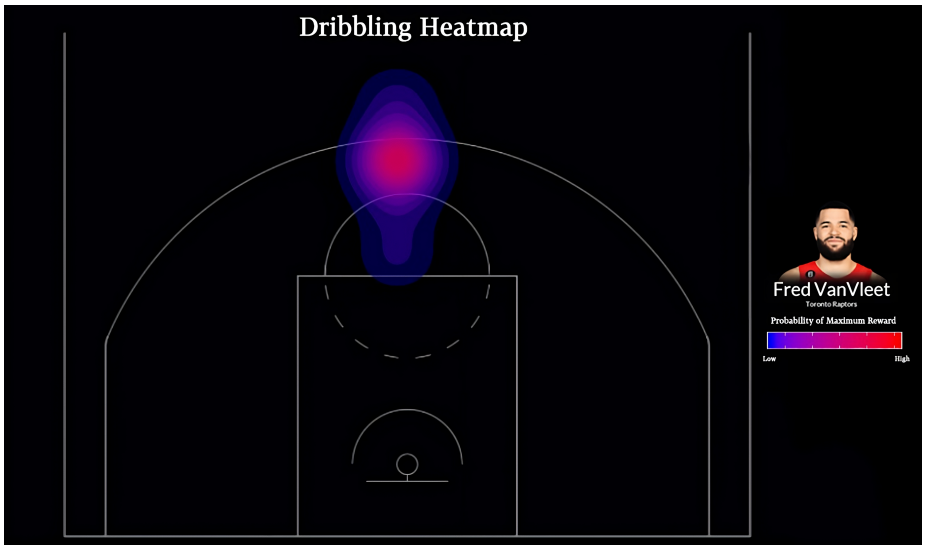


Figure 7: Fred VanVleet Heat-Map

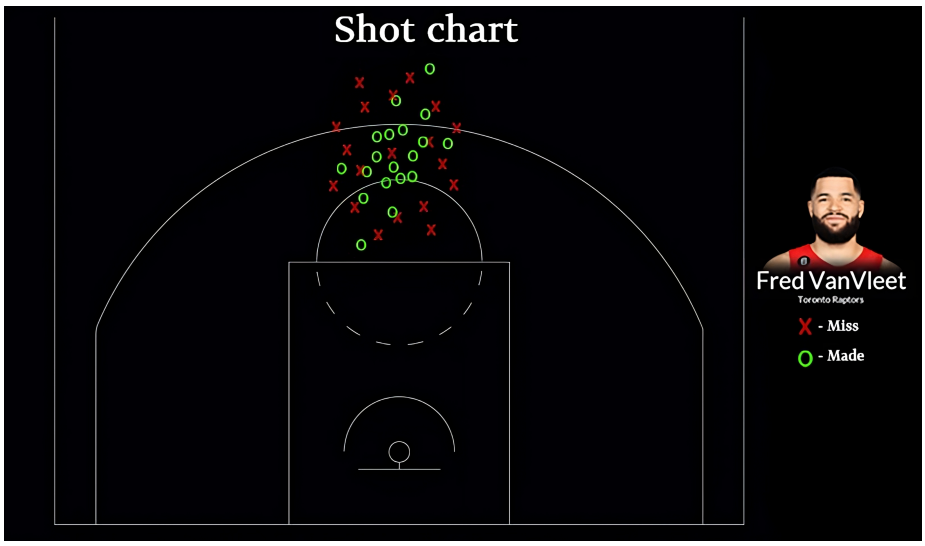


Figure 8: Fred VanVleet Shot Chart

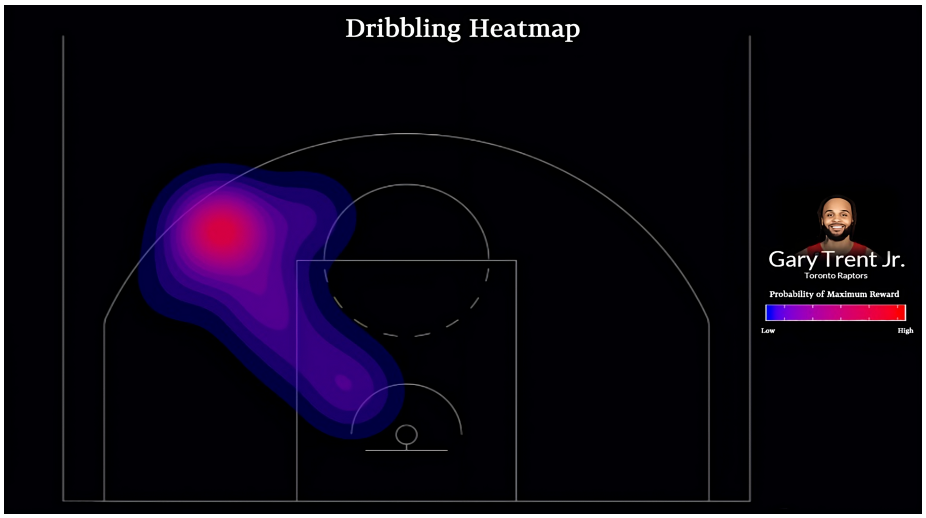


Figure 9: Gary Trent Jr. Heat-Map

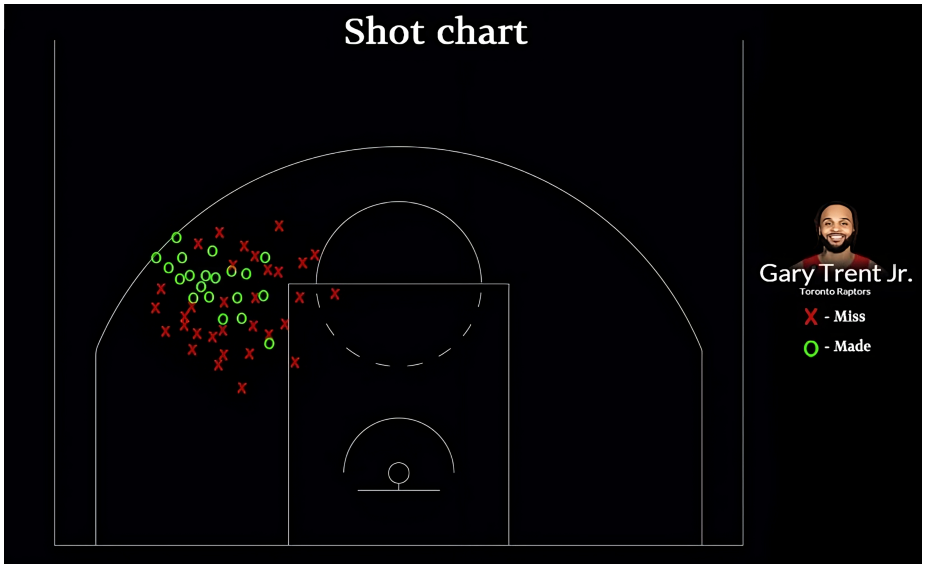


Figure 10: Gary Trent Jr. Shot Chart

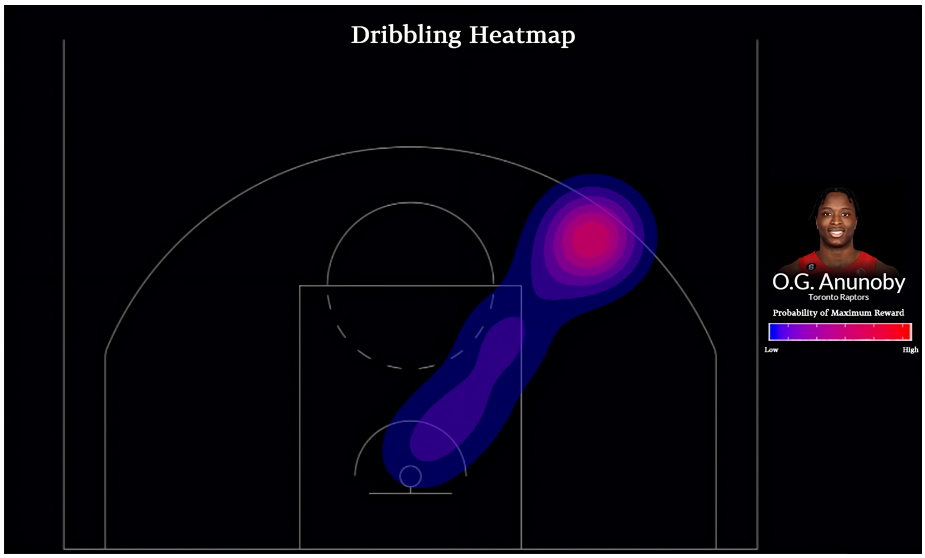


Figure 11: OG Anunoby Heat-Map

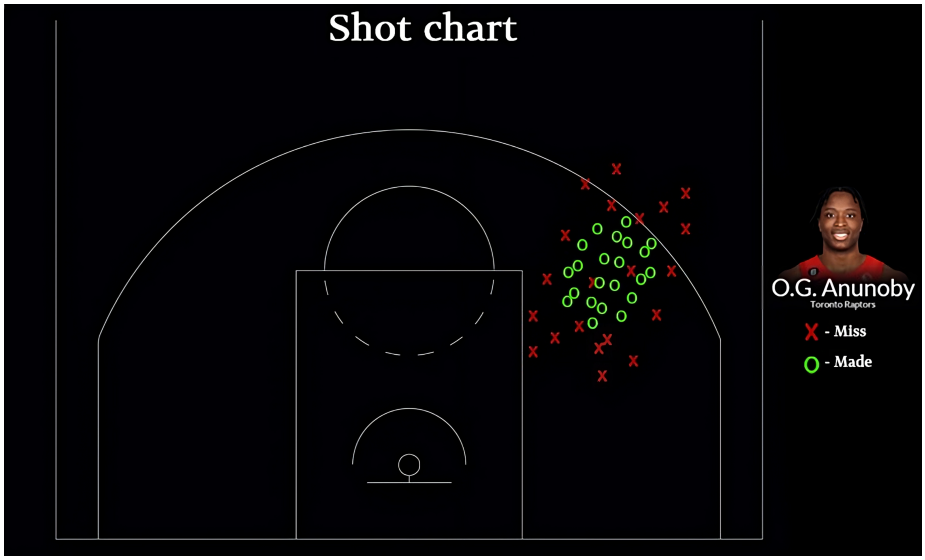


Figure 12: OG Anunoby Shot Chart

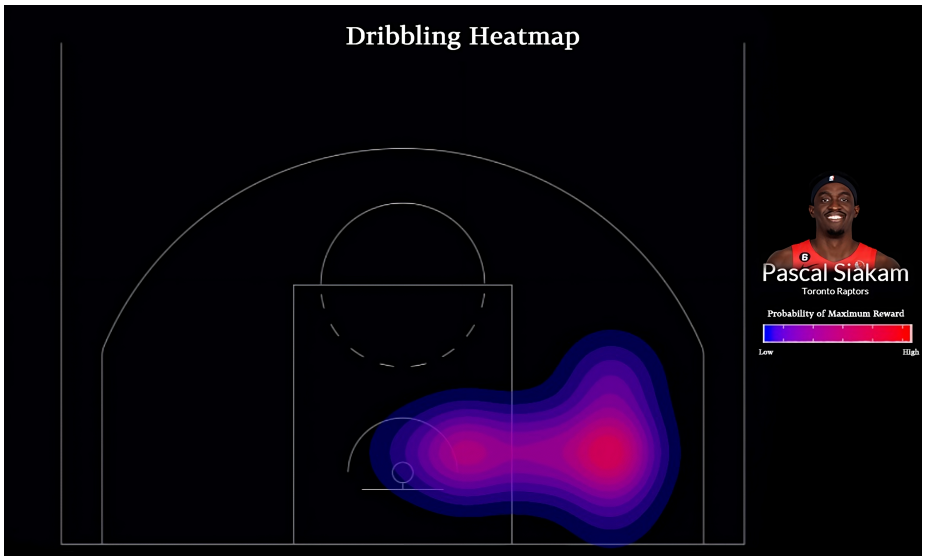


Figure 13: Pascal Siakam Heat-Map

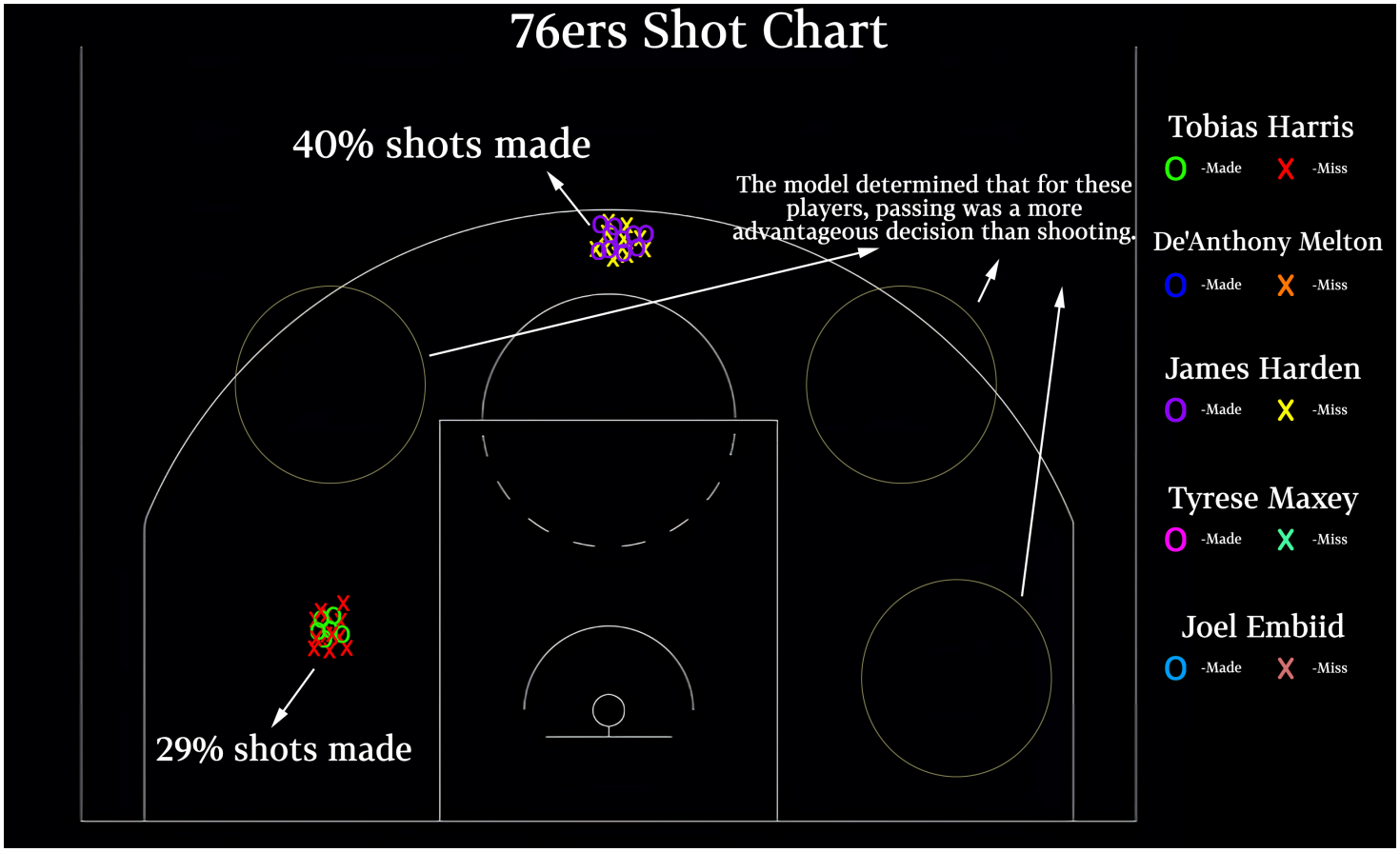


Figure 14: Philadelphia 76ers Shot-chart

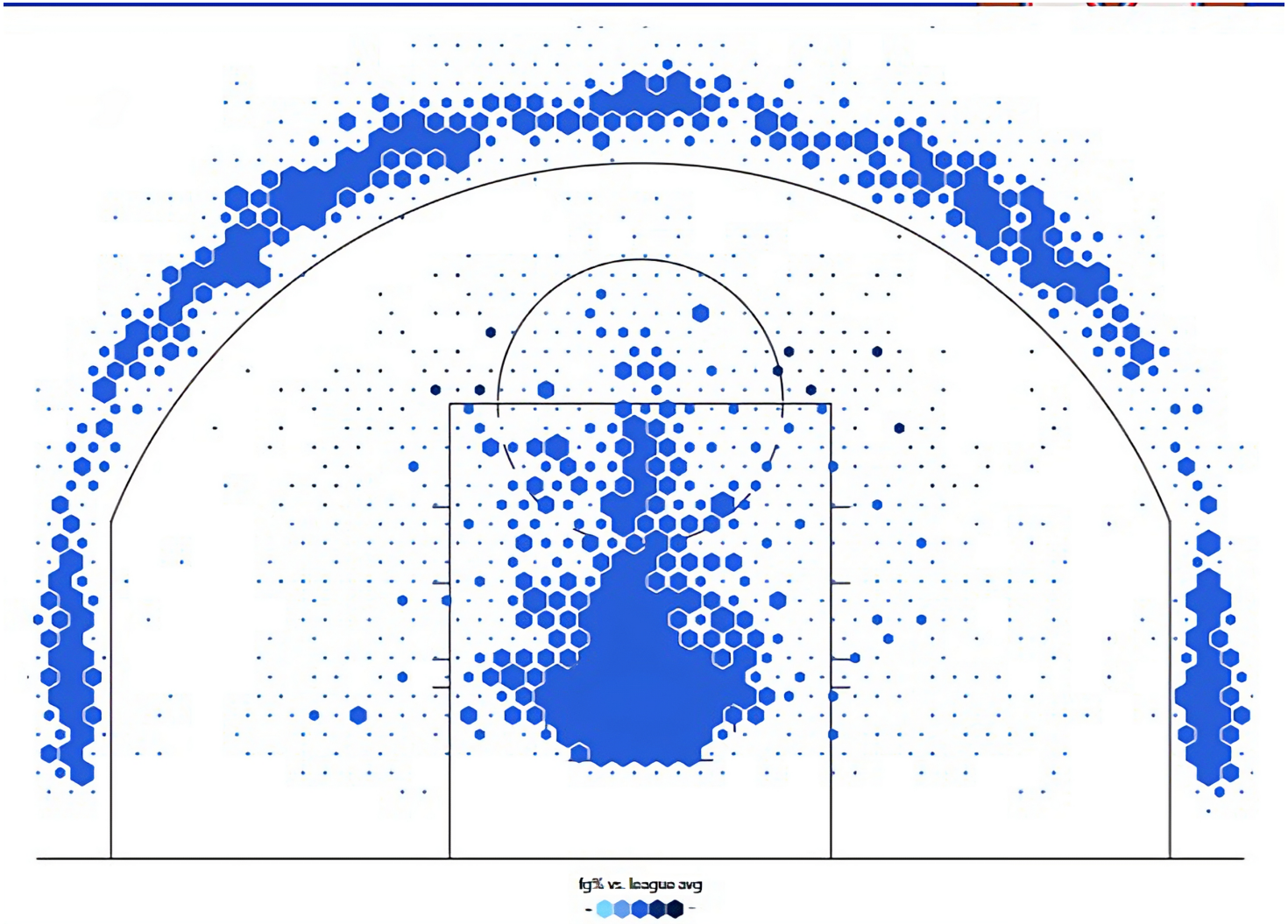


Figure 15: Philadelphia 76ers Official NBA Shot-chart
[4]

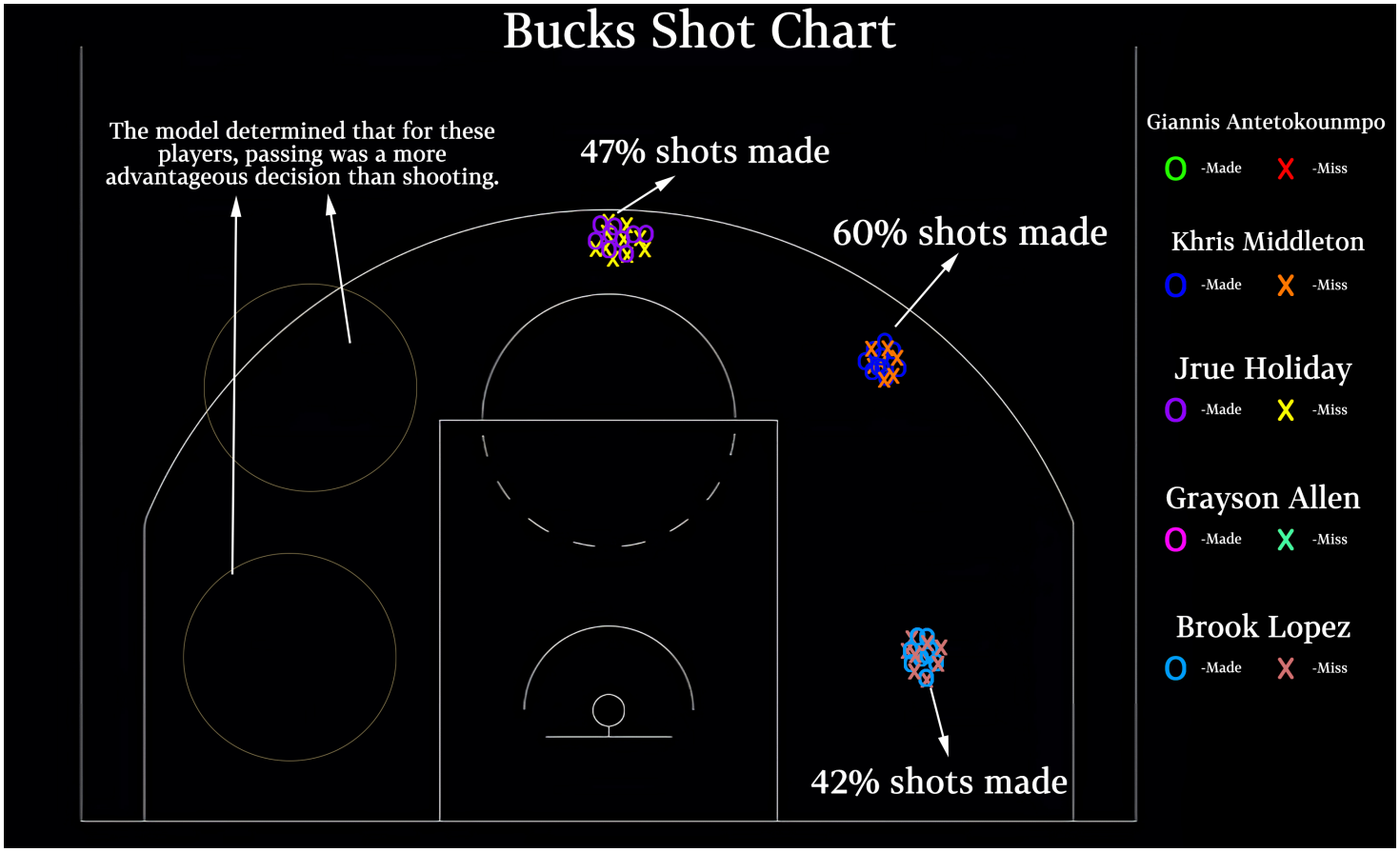


Figure 16: Milwaukee Bucks Shot Chart

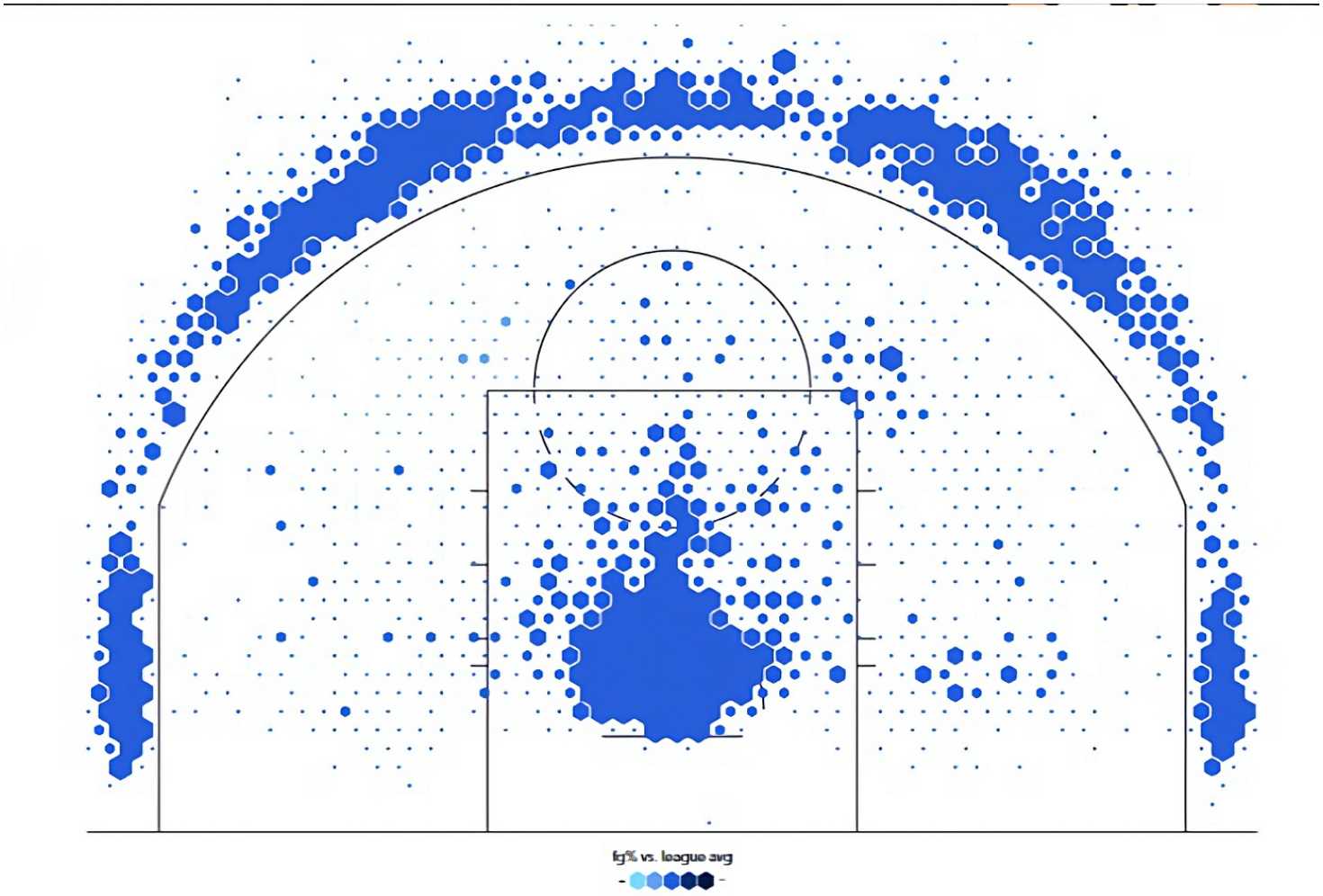


Figure 17: Milwaukee Bucks Official NBA Shot Chart
[4]

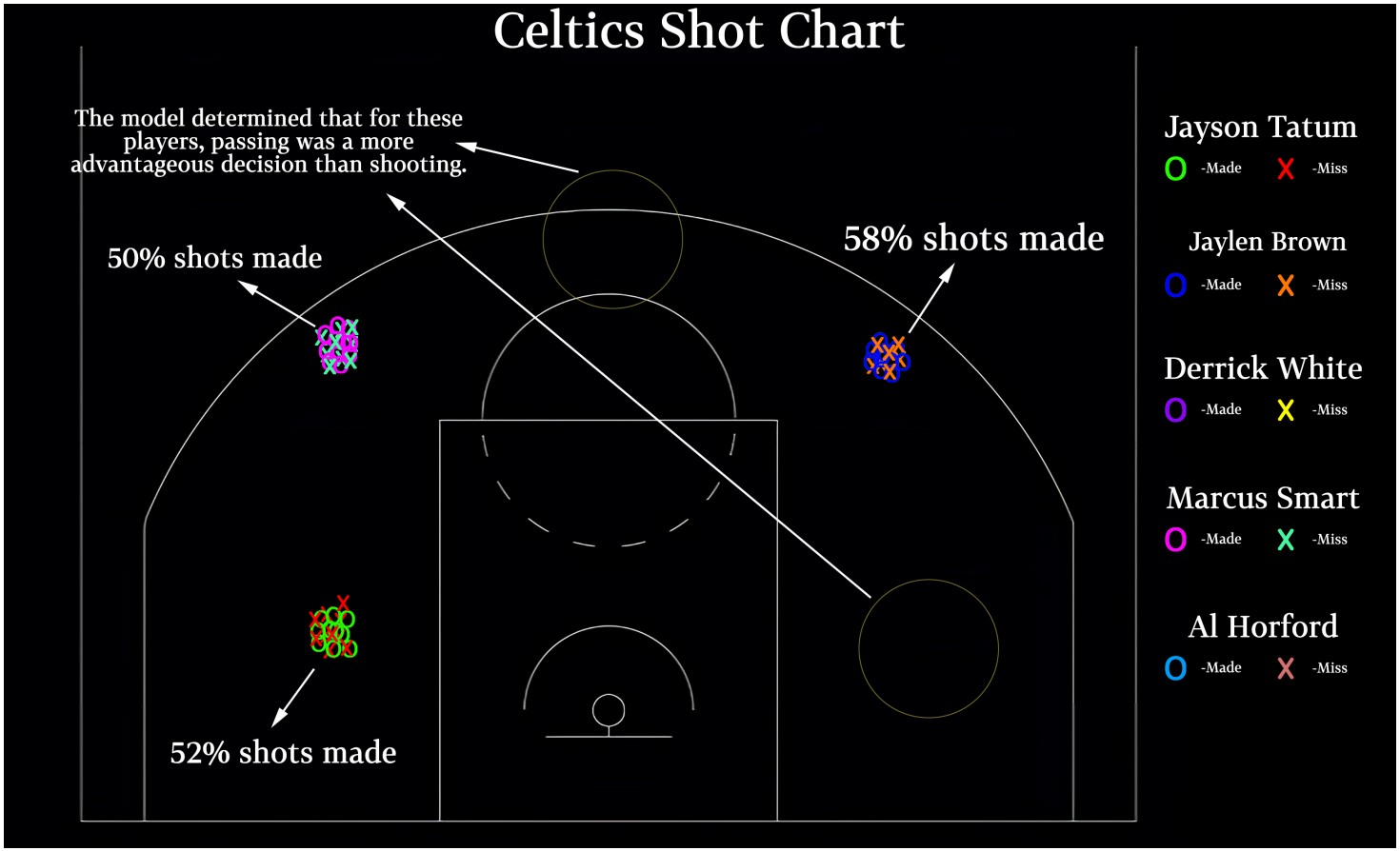


Figure 18: Boston Celtics Shot Chart

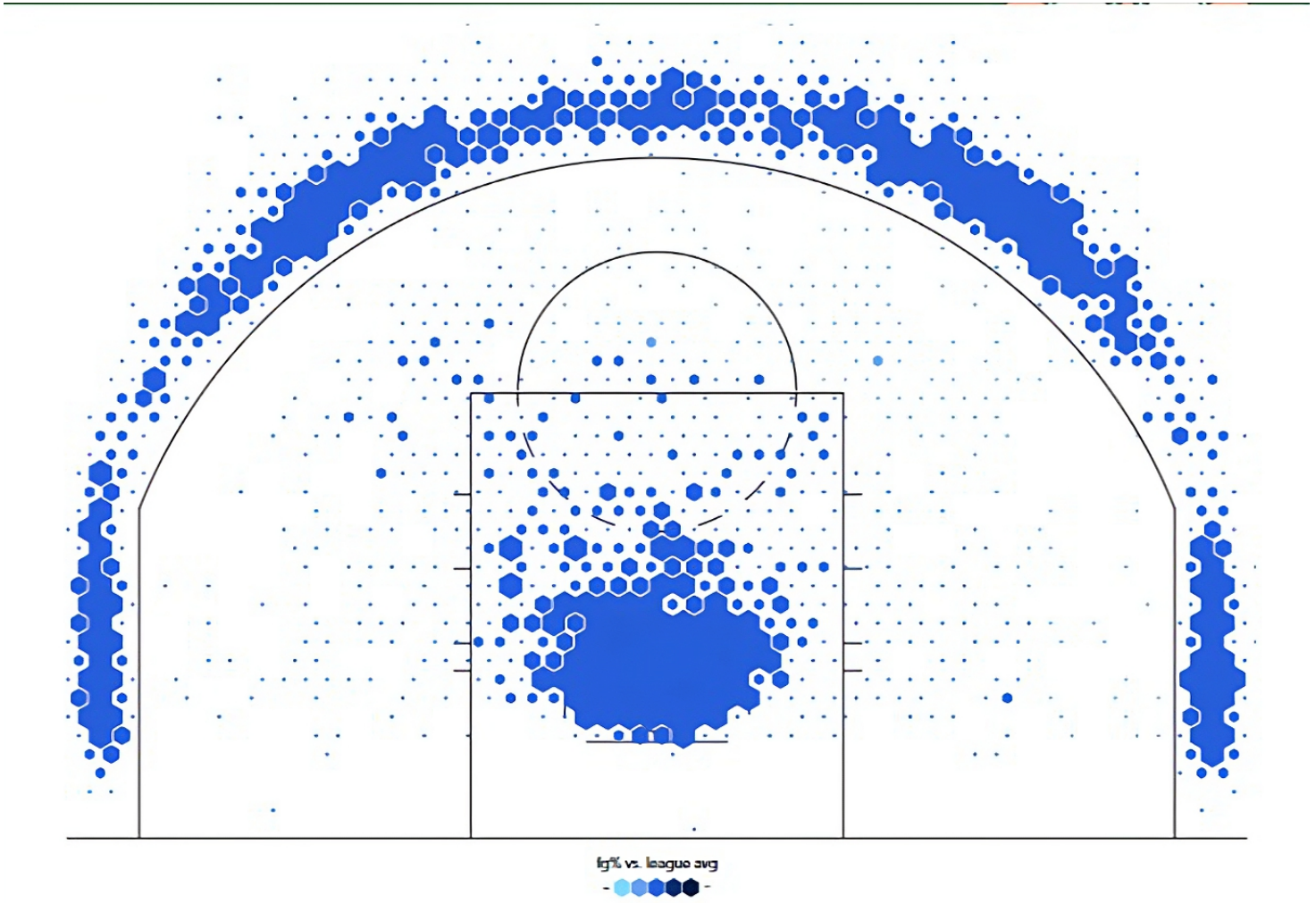


Figure 19: Boston Celtics Official NBA Shot Chart
[4]

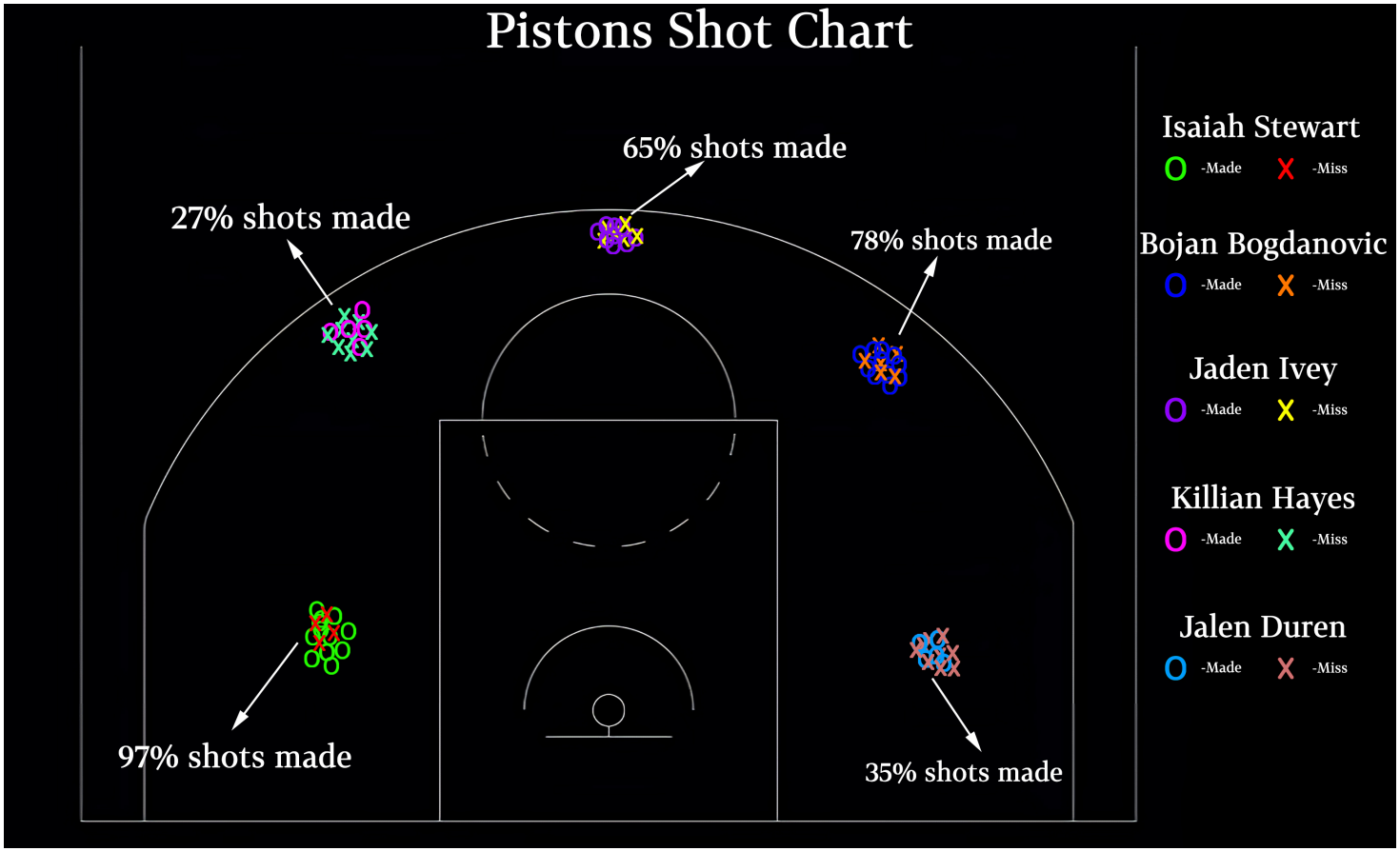


Figure 20: Detroit Pistons Shot-chart

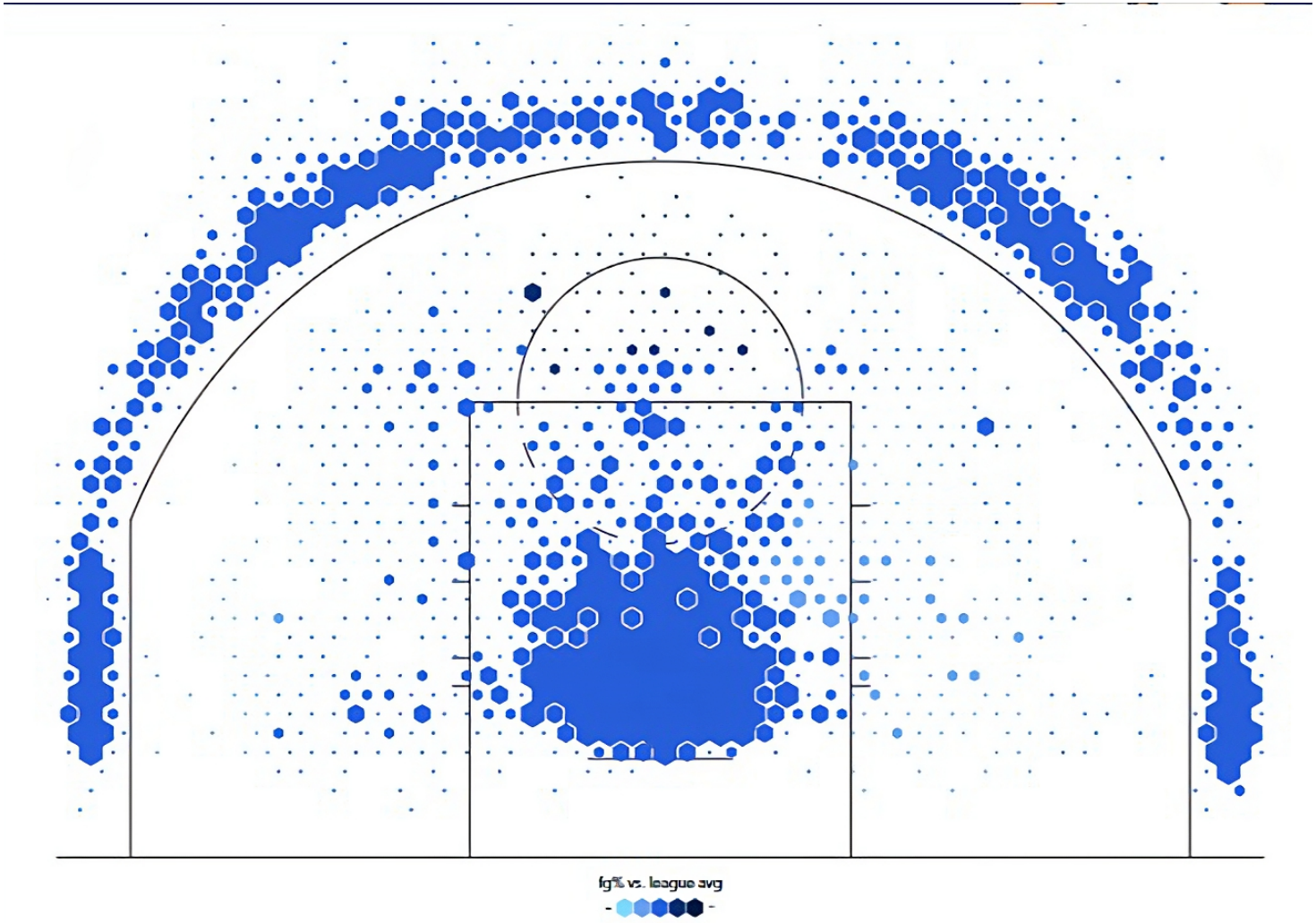


Figure 21: Detroit Pistons Official NBA Shot-chart
[4]

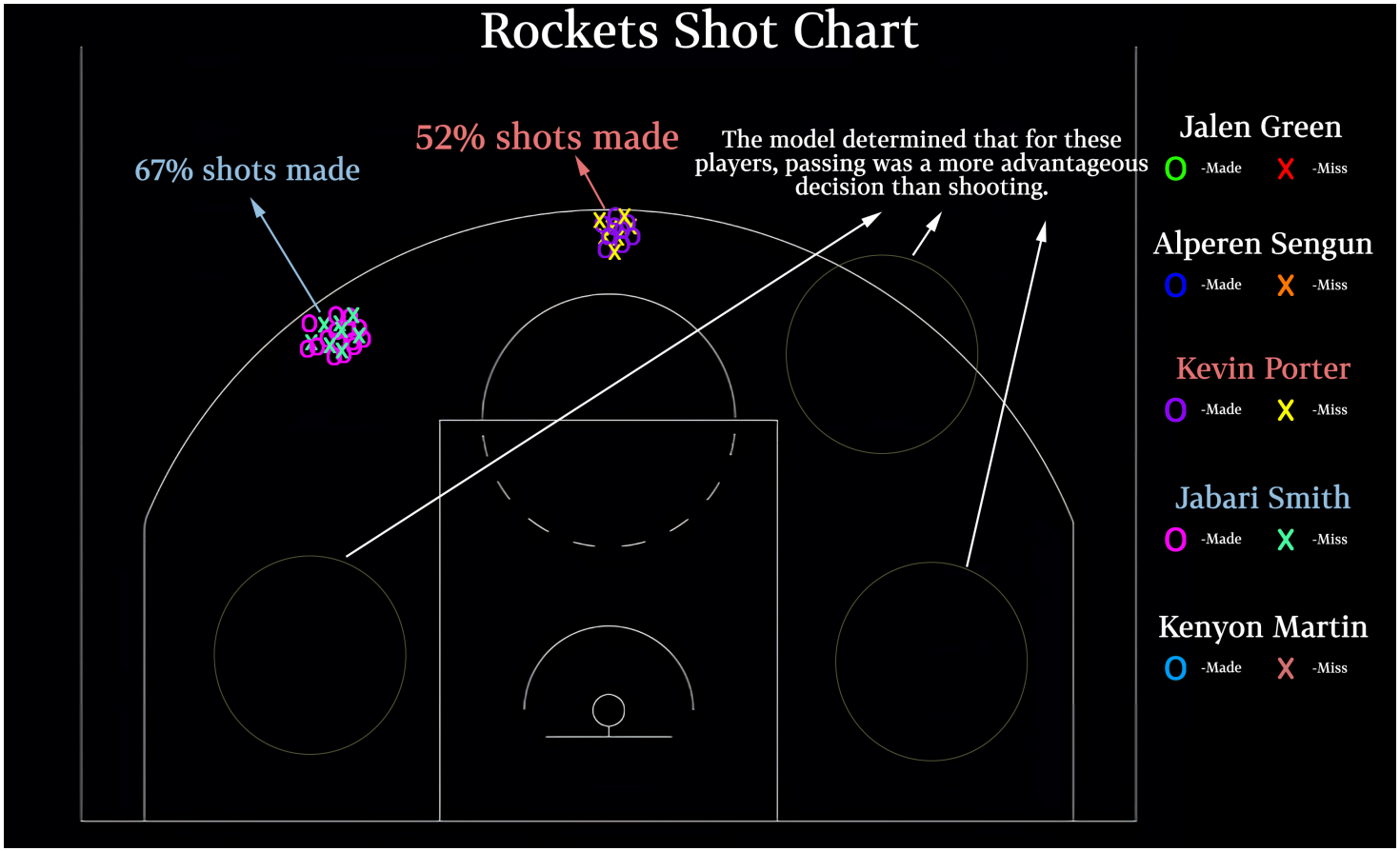


Figure 22: Houston Rockets Shot Chart

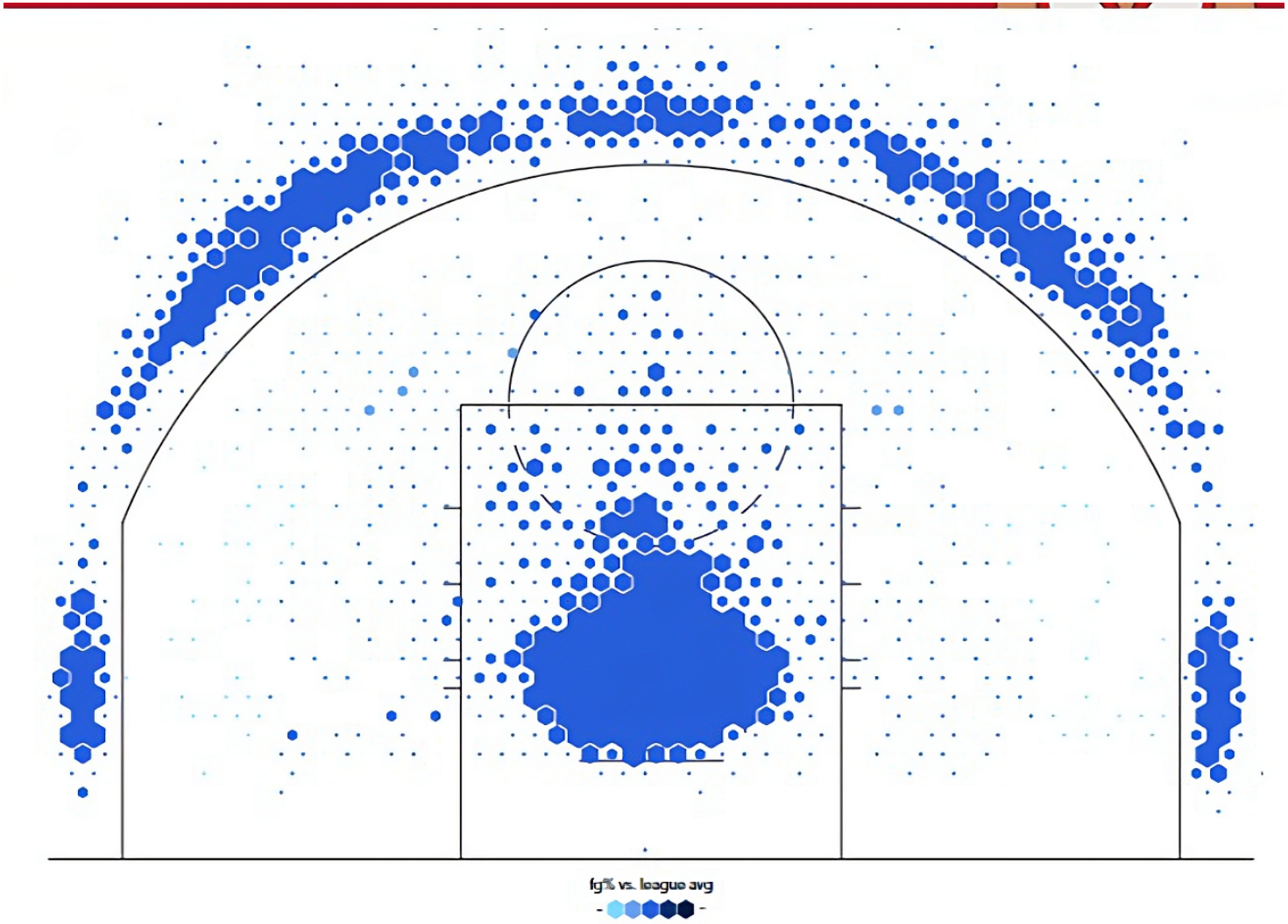


Figure 23: Houston Rockets Official NBA Shot Chart
[4]

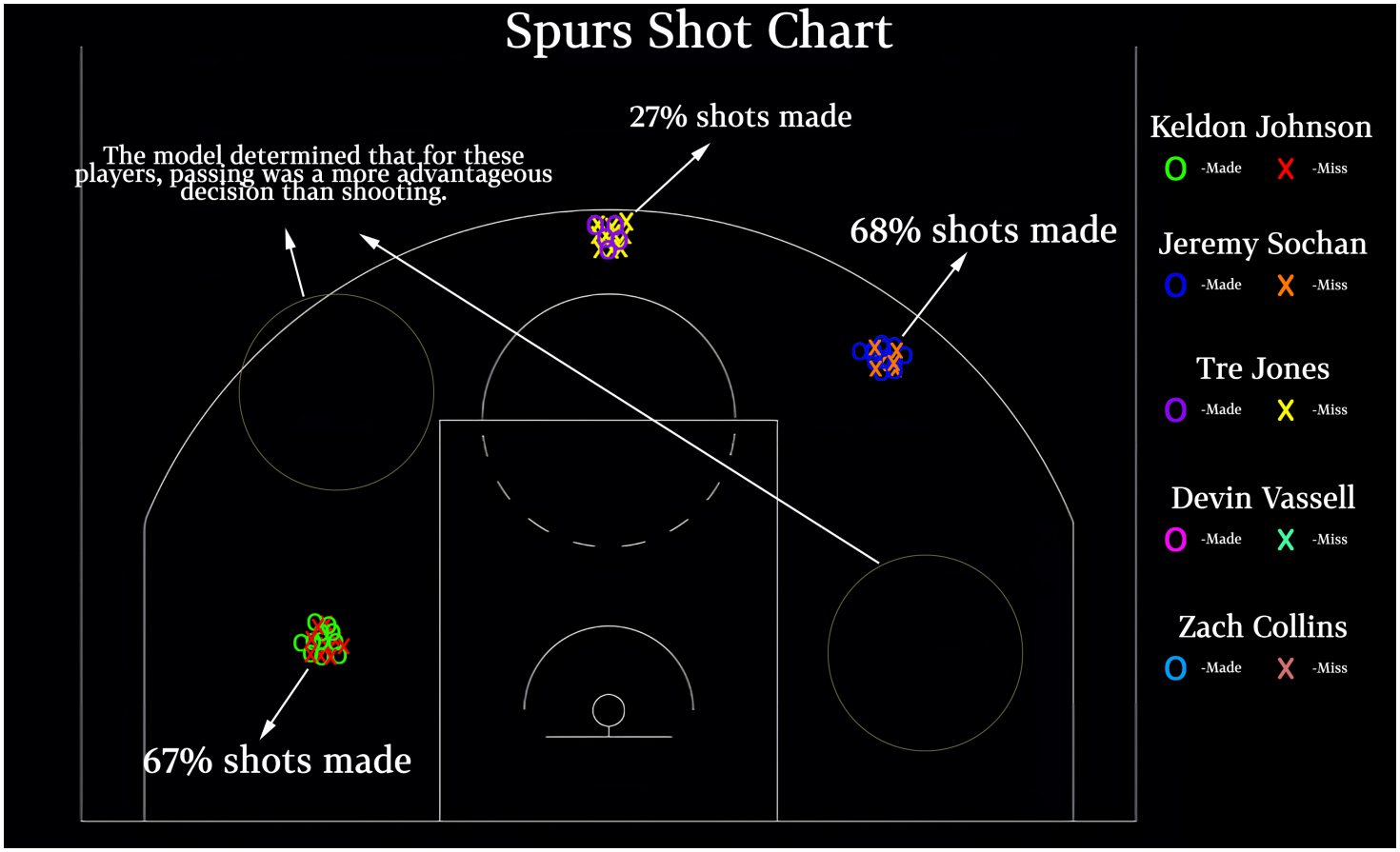


Figure 24: San Antonio Spurs Shot Chart

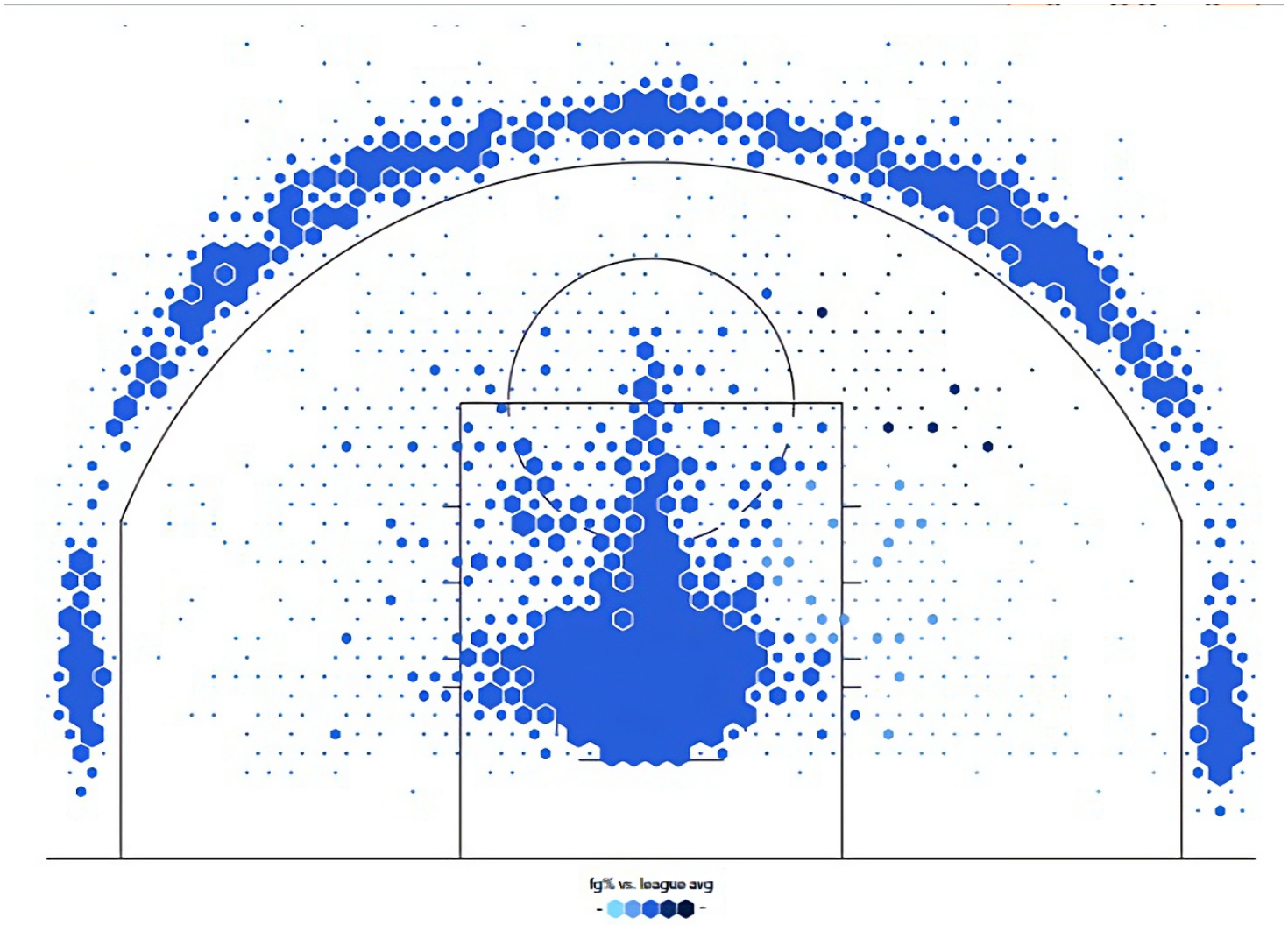


Figure 25: San Antonio Spurs Official NBA Shot Chart
[4]

References

[1] Hoopshype. https://hoopshype.com/salaries/.

[2] Nba advanced stats. https://www.nba.com/stats.

[3] Nba court optix. https://nbacourtoptix.nba.com/en/metrics/passing-network.

[4] Statmuse. https://www.statmuse.com.

[5] *Max Holloway Learned Some Striking Techniques from the UFC Video Game*. Joe Rogan Experience, Dec 2019.

[6] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games, 2018.

[7] Taylor Dohmen, Noah Topper, George Atia, Andre Beckus, Ashutosh Trivedi, and Alvaro Velasquez. Inferring probabilistic reward machines from non-markovian reward processes for reinforcement learning, 2021.

[8] Daniel Neider Jan Corazza1, Ivan Gavran. Reinforcement learning with stochastic reward machines, 2022.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[10] Michael A. Nielsen. Neural networks and deep learning, 2018.

[11] Adam Paszke. Reinforcement learning (dqn) tutorial. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.

[12] Desik Rengarajan, Gargi Vaidya, Akshay Sarvesh, Dileep Kalathil, and Srinivas Shakkottai. Reinforcement learning with sparse rewards using guidance from offline demonstration, 2022.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[14] Chen-Yu Wei, Yi-Te Hong, and Chi-Jen Lu. Online reinforcement learning in stochastic games, 2017.